Logique et démonstration automatique : Une introduction à la logique propositionnelle et à la logique du premier ordre

Version abrégée

Stéphane Devismes Pascal Lafourcade Michel Lévy

2022/2023

Ce document présente une version abrégée du cours de logique. Ainsi, la plupart des théorèmes sont admis sans démonstration, et certaines notions sont omises.
Le polycopié complet peut être retrouvé sur
https://wackb.gricad-pages.univ-grenoble-alpes.fr/inf402/.
Les références du document original ont été préservées, ce qui explique ici des « trous » dans la numérotation.

Table des matières

I Logique propositionnelle **9**

1 Logique propositionnelle **11**
1.1 Syntaxe . . . 12
1.1.1 Formules strictes . . . 12
1.1.2 Formules à priorité . . . 14
1.2 Sens des formules . . . 14
1.2.1 Sens des connecteurs . . . 14
1.2.2 Valeur d'une formule . . . 15
1.2.3 Définitions et notions élémentaires de logique . . . 15
1.2.5 Équivalences remarquables . . . 18
1.3 Substitution et remplacement . . . 19
1.3.1 Substitution . . . 19
1.3.2 Remplacement . . . 21
1.4 Formes Normales . . . 22
1.4.1 Transformation en forme normale . . . 22
1.4.2 Transformation en forme normale disjonctive (somme de monômes) . . . 23
1.4.3 Transformation en forme normale conjonctive (produit de clauses) . . . 24
1.5 Algèbre de Boole . . . 24
1.5.1 Définition et notations . . . 24
1.5.2 Propriétés . . . 25
1.6 Fonctions booléennes . . . 29
1.6.1 Fonctions booléennes et somme de monômes . . . 29
1.6.2 Fonctions booléennes et produit de clauses . . . 29
1.7 L'outil BDDC . . . 30
1.8 Exercices . . . 31

2 Résolution propositionnelle **37**
2.1 Résolution . . . 38
2.1.1 Définitions . . . 38
2.1.2 Cohérence . . . 40
2.1.3 Complétude pour la réfutation . . . 41
2.1.4 Réduction d'un ensemble de clauses . . . 42
2.2 Stratégie complète . . . 43
2.2.1 Algorithme de la stratégie complète . . . 43
2.2.2 Arrêt de l'algorithme . . . 44
2.2.3 Le résultat de l'algorithme est équivalent à l'ensemble initial de clauses . . . 45
2.3 Algorithme de Davis-Putnam-Logemann-Loveland . . . 45
2.3.1 Suppression des clauses contenant des littéraux isolés . . . 46
2.3.2 Résolution unitaire . . . 46
2.3.3 Suppression des clauses valides . . . 47
2.3.4 L'algorithme . . . 47
2.3.5 Solveurs SAT . . . 49
2.3.5.1 Heuristique de branchement . . . 50
2.3.5.2 Ajout de clauses . . . 50

2.3.5.3 Analyse des conflits et retour-arrière non chronologique . . . 50
2.3.5.4 Apprentissage . . . 50
2.3.5.5 Redémarrage . . . 50
2.3.5.6 Structures de données paresseuses . . . 51
2.4 Exercices . . . 52

3 Déduction Naturelle **57**
3.1 Système formel de la déduction naturelle . . . 58
3.1.1 Règles de conjonction . . . 58
3.1.2 Règles de disjonction . . . 58
3.1.3 Règles de l'implication . . . 59
3.1.4 Deux règles spéciales . . . 59
3.1.5 Preuves en déduction naturelle . . . 60
3.1.5.1 Brouillon de preuve . . . 60
3.1.5.2 Contexte des lignes d'un brouillon de preuve . . . 60
3.1.5.3 Preuves . . . 61
3.2 Tactiques de preuve . . . 63
3.3 Cohérence de la déduction naturelle . . . 65
3.4 Complétude de la déduction naturelle . . . 66
3.5 Outils . . . 66
3.5.1 Logiciel de construction automatique de preuves . . . 66
3.5.2 Dessiner des arbres de preuves . . . 66
3.6 Exercices . . . 67

II Logique du premier ordre **71**

4 Logique du premier ordre **73**
4.1 Syntaxe . . . 74
4.1.1 Formules strictes . . . 74
4.1.2 Formules à priorité . . . 76
4.2 Être libre ou lié . . . 77
4.2.1 Occurrences libres et liées . . . 77
4.2.2 Variables libres et liées . . . 77
4.3 Sens des formules . . . 78
4.3.1 Déclaration de symbole . . . 78
4.3.2 Signature . . . 78
4.3.3 Interprétation . . . 79
4.3.4 Sens des formules . . . 80
4.3.4.1 Sens des termes sur une signature . . . 80
4.3.4.2 Sens des formules atomiques sur une signature . . . 81
4.3.4.3 Sens des formules sur une signature . . . 81
4.3.5 Modèle, validité, conséquence, équivalence . . . 82
4.3.6 Instanciation . . . 82
4.3.7 Interprétation finie . . . 83
4.3.7.1 Les entiers et leurs représentations . . . 83
4.3.7.2 Expansion d'une formule . . . 84
4.3.7.3 Interprétation et assignation propositionnelle . . . 84
4.3.7.4 Recherche d'un modèle fini d'une formule fermée . . . 85
4.3.8 Substitution et remplacement . . . 86
4.4 Équivalences remarquables . . . 86
4.4.1 Relation entre $\forall$ et $\exists$ . . . 86
4.4.2 Déplacement des quantificateurs . . . 87
4.4.3 Changement de variables liées . . . 87
4.5 Exercices . . . 89

5 Base de la démonstration automatique **93**
5.1 Méthode de Herbrand . . . 94
5.1.1 Domaine et base de Herbrand . . . 94
5.1.2 Interprétation de Herbrand . . . 94
5.1.3 Théorème de Herbrand . . . 95
5.2 Skolémisation . . . 97
5.2.1 Algorithme de skolémisation d'une formule . . . 97
5.2.1.1 Transformation en formule normale . . . 98
5.2.1.2 Transformation en formule propre . . . 98
5.2.1.3 Élimination des quantificateurs existentiels . . . 98
5.2.1.4 Transformation en fermeture universelle . . . 99
5.2.2 Propriétés de la forme de Skolem . . . 99
5.2.3 Forme clausale . . . 101
5.3 Unification . . . 102
5.3.1 Unificateur . . . 102
5.3.2 Algorithme d'unification . . . 103
5.3.2.1 Les règles de l'algorithme . . . 104
5.3.2.2 Correction de l'algorithme . . . 105
5.3.2.3 Terminaison de l'algorithme . . . 105
5.4 Résolution au premier ordre . . . 105
5.4.1 Trois règles pour la résolution . . . 105
5.4.1.1 Factorisation . . . 106
5.4.1.2 Copie d'une clause . . . 106
5.4.1.3 Résolution binaire . . . 107
5.4.1.4 Preuve par factorisation, copie et résolution binaire . . . 107
5.4.2 Cohérence de la résolution . . . 107
5.4.3 Complétude de la résolution . . . 108
5.5 Outil logiciel . . . 110
5.6 Exercices . . . 111

6 Déduction naturelle au premier ordre : quantificateurs, copie et égalité **115**
6.1 Règles pour la logique du premier ordre . . . 115
6.1.1 Règles des quantificateurs . . . 116
6.1.1.1 Règles pour le quantificateur universel . . . 116
6.1.1.2 Règles pour le quantificateur existentiel . . . 117
6.1.2 Copie . . . 119
6.1.3 Les règles de l'égalité . . . 119
6.2 Tactiques de preuves . . . 119
6.2.1 Raisonner en avant avec une hypothèse d'existence . . . 120
6.2.2 Raisonner en arrière pour généraliser . . . 120
6.2.3 Un exemple d'application des tactiques . . . 120
6.3 Cohérence du système . . . 122
6.4 Exercices . . . 124

Bibliographie **127**

Introduction

LES mathématiciens ont raisonné *correctement* durant des siècles *sans connaître* la logique formelle. La logique moderne est née de l'ambition de formaliser (mécaniser) le raisonnement mathématique : ce projet a reçu une impulsion décisive quand il est apparu que l'absence de formalisation pouvait conduire à des contradictions. Le développement actuel de la logique continue avec

— le besoin de prouver la correction des programmes (particulièrement quand ces programmes sont utilisés dans des domaines où la sécurité est en jeu)

— l'ambition de représenter en machine l'ensemble des connaissances mathématiques

Cependant la terminologie des mathématiciens est restée différente de celle des logiciens. Pour exprimer que p implique q, le mathématicien dira par exemple :

p entraîne q
p est une condition suffisante de q
pour que q soit vrai, il suffit que p soit vrai
q est une condition nécessaire de p
pour que p soit vrai, il faut que q soit vrai

Nous restreindrons notre étude à la logique *classique* (par opposition à la logique intuitionniste). La logique classique est la logique à deux valeurs de vérité. De plus cette logique est celle des circuits combinatoires, ce qui explique sa grande importance pratique.

Plan : Nous présentons dans la première partie la logique propositionnelle. Plus précisément dans un premier chapitre nous donnons les définitions et résultats de base de la logique des prédicats. Dans le second chapitre nous parlons de la résolution propositionnelle en introduisant la résolution binaire, la stratégie complète et l'algorithme `DPLL`. Enfin nous illustrons une méthode de raisonnement logique en présentant la déduction naturelle. Dans le seconde partie du cours nous revisitons l'ensemble des notions, résultats et techniques présentés dans la première partie pour la logique du premier ordre.

Exercices : Nous proposons à la fin de chaque chapitre une série d'exercices portant sur le contenu du chapitre. Nous indiquons par des étoiles la difficulté des exercices proposés, plus il y a d'étoiles plus l'exercice est difficile. Nous indiquons par ⟲ les exercices qui complètent les preuves vues durant le cours. Les exercices proposés sont de trois catégories. Nous avons des exercices :

— Basiques qui aident l'étudiant à se familiariser avec le vocabulaire et à manipuler les notions introduites en cours.

— Techniques qui sont des applications directes ou immédiates des résultats vus en cours.

— Réflexifs qui permettent à l'apprenant à raisonner, démontrer, prouver des résultats à partir des techniques et notions apprises grâce aux deux autres types d'exercices.

Objectifs : Les compétences et connaissances que nous souhaitons transmettre sont les suivantes :

— *Comprendre un raisonnement* : être capable de déterminer si un raisonnement logique est correct ou non.

— *Raisonner*, c'est-à-dire, construire un raisonnement correct utilisant les outils de la logique propositionnelle et du premier ordre.

— *Modéliser et formaliser un problème*.

— *Écrire une preuve rigoureuse*.

Première partie

Logique propositionnelle

Chapitre 1

Logique propositionnelle

Sommaire

1.1 Syntaxe **12**
1.1.1 Formules strictes 12
1.1.2 Formules à priorité 14
1.2 Sens des formules **14**
1.2.1 Sens des connecteurs 14
1.2.2 Valeur d'une formule 15
1.2.3 Définitions et notions élémentaires de logique 15
1.2.5 Équivalences remarquables 18
1.3 Substitution et remplacement **19**
1.3.1 Substitution 19
1.3.2 Remplacement 21
1.4 Formes Normales **22**
1.4.1 Transformation en forme normale 22
1.4.2 Transformation en forme normale disjonctive (somme de monômes) 23
1.4.3 Transformation en forme normale conjonctive (produit de clauses) 24
1.5 Algèbre de Boole **24**
1.5.1 Définition et notations 24
1.5.2 Propriétés 25
1.6 Fonctions booléennes **29**
1.6.1 Fonctions booléennes et somme de monômes 29
1.6.2 Fonctions booléennes et produit de clauses 29
1.7 L'outil BDDC **30**
1.8 Exercices **31**

ARISTOTE fut un des premiers à essayer de formaliser le raisonnement en utilisant la logique des syllogismes. La logique sert à préciser ce qu'est un raisonnement correct, indépendamment du domaine d'application. Un raisonnement est un moyen d'obtenir une conclusion à partir d'hypothèses données. Un raisonnement *correct* ne dit rien sur la vérité des hypothèses, il dit seulement qu'à partir de *la vérité des hypothèses, nous pouvons déduire la vérité de la conclusion*. Nous commençons l'étude de la logique par les lois de la logique propositionnelle. La logique propositionnelle est la logique *sans quantificateurs* qui s'intéresse uniquement aux lois gouvernant les opérations logiques suivantes : la négation ($\neg$) , la conjonction, autrement dit le « et » ($\wedge$), la disjonction, autrement dit le « ou » ($\vee$), l'implication ($\Rightarrow$) et l'équivalence ($\Leftrightarrow$). Ces opérations sont également appelées *connecteurs*. La logique propositionnelle permet de construire des raisonnements à partir de ces connecteurs. Considérons l'exemple suivant qui comporte trois hypothèses :

1. si Pierre est grand, alors Jean n'est pas le fils de Pierre,
2. si Pierre n'est pas grand, alors Jean est le fils de Pierre,
3. si Jean est le fils de Pierre alors Marie est la sœur de Jean.

Nous concluons que Marie est la sœur de Jean ou Pierre est grand.

Afin de pouvoir raisonner nous extrayons la structure logique des hypothèses. Nous désignons les phrases « Pierre est grand », « Jean est le fils de Pierre », « Marie est la sœur de Jean » respectivement par les lettres p, j, m. Les hypothèses peuvent donc s'écrire :

1. $p \Rightarrow \neg j$,
2. $\neg p \Rightarrow j$,
3. $j \Rightarrow m$,

et la conclusion se formalise en $m \vee p$. Nous montrons alors que les hypothèses impliquent la conclusion indépendamment de la nature des énoncés p, j, m. Pour cela nous prouvons que la formule suivante est vraie quelle que soit la vérité des propositions p, j, m.

$$((p \Rightarrow \neg j) \wedge (\neg p \Rightarrow j) \wedge (j \Rightarrow m)) \Rightarrow (m \vee p).$$

Plan : Nous débutons ce chapitre par la *syntaxe des formules logiques*, c'est-à-dire, les règles permettant d'écrire des formules. Une formule peut être vraie ou fausse, ainsi nous devons être capable d'évaluer le *sens d'une formule*. Pour cela nous introduisons le sens de chaque connecteur ainsi que le calcul de la valeur d'une formule qui en dérive. Nous montrons alors un résultat de compacité qui sera principalement utilisé dans la seconde partie de ce livre. Ensuite nous présentons des *équivalences remarquables* utiles pour simplifier les raisonnements logiques. D'autres méthodes permettent de simplifier les raisonnements logiques, par exemple le *remplacement* et la *substitution* de formule. Nous montrons ensuite comment construire les formes normales conjonctives ou disjonctives d'une formule en utilisant les équivalences remarquables. Ces formes normales permettent d'exhiber facilement les modèles ou les contre-modèles d'une formule. Nous montrons ensuite que la logique propositionnelle est une instance d'une *algèbre de Boole*. Nous introduisons la notion de *fonctions booléennes*. Enfin, nous présentons succinctement l'outil `BDDC`[1] développé par Pascal Raymond. Cet outil permet de manipuler les formules propositionnelles.

1.1 Syntaxe

Avant de raisonner, nous définissons le langage que nous utilisons. Ce langage est celui des formules construites à partir du *vocabulaire* suivant :

— Les constantes : $\top$ et $\bot$ représentant respectivement le *vrai* et le *faux*.
— Les variables : une variable est un identificateur, avec ou sans indice, par exemple x, y_1.
— Les parenthèses : ouvrante (et fermante).
— Les connecteurs : $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$ respectivement appelés négation, disjonction (ou), conjonction (et), implication et équivalence.

La syntaxe définit les règles de construction d'une formule de la logique propositionnelle. Nous introduisons deux modes d'écriture d'une formule, l'un strict, l'autre plus souple dans le sens qu'il autorise plusieurs écritures d'une même formule. Cette souplesse est obtenue grâce à l'introduction de priorités entre les connecteurs logiques.

1.1.1 Formules strictes

Ci-dessous, nous donnons les règles de construction d'une formule stricte à partir du vocabulaire donné précédemment.

Définition 1.1.1 (Formule stricte) *Une* formule stricte *est définie de manière inductive comme suit :*

— *$\top$ et $\bot$ sont des formules strictes.*
— *Une variable est une formule stricte.*
— *Si A est une formule stricte alors $\neg A$ est une formule stricte.*
— *Si A et B sont des formules strictes et si $\circ$ est une des opérations $\vee, \wedge, \Rightarrow, \Leftrightarrow$ alors $(A \circ B)$ est une formule stricte.*

Dans la suite, nous désignons par $\circ$ tout connecteur binaire, et nous appellons simplement formule une formule stricte. Les formules différentes de $\top$, $\bot$ et des variables sont des formules décomposables.

Exemple 1.1.2 *L'expression $(a \vee (\neg b \wedge c))$ est une formule stricte construite suivant les règles précédentes. En revanche, $a \vee (\neg b \wedge c)$ et $(a \vee (\neg (b) \wedge c))$ ne sont pas des formules au sens de la définition 1.1.1.*

1. `http://www-verimag.imag.fr/~raymond/home/tools/bddc/`.

L'intérêt de la définition de formules strictes est que les parenthèses permettent de trouver sans ambiguïté la structure des formules. Nous représentons la structure des formules par un arbre, où les feuilles contiennent les constantes ou les variables et les nœuds les connecteurs logiques. Le nœud racine est le connecteur à appliquer en dernier.

Exemple 1.1.3 *La structure de la formule* $(a \vee (\neg b \wedge c))$ *est mise en évidence par l'arbre suivant :*

Une formule peut être vue comme une liste de symboles (connecteurs, parenthèses, variables, et constantes). Un facteur d'une telle liste est une suite de symboles consécutifs dans la liste.

Définition 1.1.4 (Sous-formule) *Nous appelons* sous-formule *d'une formule (stricte) A tout facteur de A qui est une formule (stricte).*

Exemple 1.1.5 $(\neg b \wedge c)$ *est une sous-formule de* $(a \vee (\neg b \wedge c))$.

Nous montrons que les formules sont décomposables d'une façon unique en leurs sous-formules. Ce résultat, précisé par le théorème 1.1.13, est évident sur les exemples. L'unicité de la décomposition implique que nous pouvons identifier une formule et son arbre de décomposition. Ainsi, une sous-formule de la formule A pourra être identifiée comme un sous-arbre de l'arbre représentant la formule A.

Afin de raisonner par induction sur la structure d'une formule, nous définissons la taille d'une formule. Nous remarquons que la taille d'une formule correspond au nombre de connecteurs qu'elle contient.

Définition 1.1.10 (Taille d'une formule) *La* taille d'une formule A, *notée* $|A|$, *est définie inductivement par :*

— $|\top| = 0$ *et* $|\bot| = 0$.
— *Si A est une variable alors* $|A| = 0$.
— $|\neg A| = 1 + |A|$.
— $|(A \circ B)| = |A| + |B| + 1$.

Exemple 1.1.11 $|(a \vee (\neg b \wedge c))| =$

La *taille* des formules définie dans la définition 1.1.10 est une mesure utile pour prouver par récurrence des propriétés sur les formules.

Théorème 1.1.13 *Pour toute formule A, un et un seul de ces cas se présente :*

— *A est une variable,*
— *A est une constante,*
— *A s'écrit d'une unique façon sous la forme $\neg B$ où B est une formule,*
— *A s'écrit d'une unique façon sous la forme $(B \circ C)$ où B et C sont des formules.*

Avec la définition de formules (strictes) nous écrivons de nombreuses parenthèses inutiles comme les parenthèses qui entourent chaque formule. Nous introduisons maintenant plus de souplesse dans notre syntaxe en définissant des priorités.

1.1.2 Formules à priorité

Pour éviter la surabondance des parenthèses, nous définissons les *formules à priorité*.

Définition 1.1.14 (Formule à priorité) *Une* formule à priorité *est définie inductivement par :*
- $\top$ *et* $\bot$ *sont des formules à priorité,*
- *une variable est une formule à priorité,*
- *si* A *est une formule à priorité alors* $\neg A$ *est une formule à priorité,*
- *si* A *et* B *sont des formules à priorité alors* $A \circ B$ *est une formule à priorité,*
- *si* A *est une formule à priorité alors* (A) *est une formule à priorité.*

Exemple 1.1.15 *Considérons la formule* $a \vee \neg b \wedge c$ *qui est une formule à priorité mais pas une formule.*

En général, une formule à priorité n'est pas une formule (stricte). Nous montrons dans l'exercice 2 page 31 que toute formule est une formule à priorité. Afin de pouvoir supprimer des parenthèses sans aucune ambiguïté nous définissons un ordre de priorité entre les différents connecteurs.

Définition 1.1.16 (Ordre de priorité des connecteurs) *La négation est prioritaire, puis dans l'ordre des priorités décroissantes, nous trouvons la conjonction* ($\wedge$)*, la disjonction* ($\vee$)*, l'implication* ($\Rightarrow$) *et l'équivalence* ($\Leftrightarrow$)*.*

À priorité égale, le connecteur gauche est prioritaire, **sauf pour l'implication** *qui est associative à droite.*[2]

Nous considérons qu'une formule à priorité est *l'abréviation* de la formule reconstituable en utilisant les priorités. Sauf exception, nous identifions une formule et son abréviation. Autrement dit, ce qui nous intéresse dans une formule, ce n'est pas son écriture *superficielle*, c'est sa structure, qui est mise en évidence par la syntaxe « stricte ». Ainsi la taille d'une formule à priorité sera égale à la taille de la formule stricte dont elle est l'abréviation. De même, pour les sous-formules, nous considérerons toujours la formule stricte dont la formule à priorité est l'abréviation.

Exemple 1.1.17 *Nous donnons plusieurs exemples d'abréviation de formule par une formule à priorité :*
- $a \wedge b \wedge c$ *est l'abréviation de*
- $a \wedge b \vee c$ *est l'abréviation de*
- $a \vee b \wedge c$ *est l'abréviation de*

Maintenant que la syntaxe est définie, nous définissons le sens des formules.

1.2 Sens des formules

Nous cherchons à déterminer si une formule est vraie ou fausse indépendamment des valeurs affectées à ses variables. Nous définissons d'abord le sens des connecteurs logiques. Ensuite nous expliquons comment calculer la valeur d'une formule et montrons le théorème de compacité. Nous terminons cette section par la présentation de définitions de notions de base de la logique qui constituent le langage commun des logiciens.

1.2.1 Sens des connecteurs

Nous désignons les valeurs de vérité par 0 pour faux et par 1 pour vrai. La constante $\top$ vaut 1 et la constante $\bot$ vaut 0, ce qui nous conduit, le plus souvent, à confondre les constantes et leurs valeurs, et à utiliser indifféremment $\top$, 1 et vrai, respectivement $\bot$, 0 et faux. Le sens des connecteurs logiques est donné par la table 1.1 page ci-contre qui indique les valeurs des formules de la première ligne suivant les valeurs *assignées* aux variables x et y.

2. C'est-à-dire $a \Rightarrow b \Rightarrow c$ est l'abréviation de $(a \Rightarrow (b \Rightarrow c))$.

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \Rightarrow y$	$x \Leftrightarrow y$
0	0					
0	1					
1	0					
1	1					

TABLE 1.1 – Table de vérité des connecteurs.

1.2.2 Valeur d'une formule

Chacun sait évaluer une formule : nous associons à chaque variable de la formule une valeur dans l'ensemble $\mathbb{B} = \{0,1\}$. La valeur de la formule est obtenue en remplaçant les variables par leurs valeurs et en effectuant les opérations suivant la table 1.1. Néanmoins, pour raisonner sur les formules, nous définissons formellement la valeur d'une formule.

Définition 1.2.1 (Assignation) *Une* assignation *est une application de l'ensemble de toutes les variables d'une formule dans l'ensemble* $\mathbb{B}$.

Définition 1.2.2 (Valeur d'une formule) *Soient A une formule et v une assignation, $[A]_v$ dénote la valeur de la formule A dans l'assignation v. La* valeur *de $[A]_v$ est définie par récurrence sur l'ensemble des formules. Soient A, B des formules, x une variable et v une assignation.*

- $[x]_v = v(x)$.
- $[\top]_v = 1$, $[\bot]_v = 0$.
- $[\neg A]_v = 1 - [A]_v$ *autrement dit pour calculer la valeur de $\neg A$, nous soustrayons à 1 la valeur de A.*
- $[(A \vee B)]_v = \max\{[A]_v, [B]_v\}$.
- $[(A \wedge B)]_v = \min\{[A]_v, [B]_v\}$.
- $[(A \Rightarrow B)]_v =$ *si* $[A]_v = 0$ *alors* 1 *sinon* $[B]_v$.
- $[(A \Leftrightarrow B)]_v =$ *si* $[A]_v = [B]_v$ *alors* 1 *sinon* 0.

D'après le théorème 1.1.13 page 13, toute formule (stricte) se décompose de façon unique en l'un des cas ci-dessus. Ainsi l'extension de v aux formules est une application des formules dans $\mathbb{B}$. En effet, soient 4 formules A, A', B, B' et deux opérations $\circ$ et $\circ'$ telles que $(A \circ B) = (A' \circ' B')$. Par unicité de la décomposition, $A = A', B = B', \circ = \circ'$, donc la valeur de la formule $(A \circ B)$ est définie uniquement par une et une seule des lignes de la définition de la valeur. Il est clair que la valeur d'une formule ne dépend que de ses variables et de sa structure, aussi l'évaluation d'une formule est présentée sous la forme d'une *table de vérité*.

Définition 1.2.3 (Table de vérité d'une formule) *Une* table de vérité d'une formule *A est un tableau qui représente la valeur de A pour toutes les valeurs possibles des variables de A.*

Chaque ligne de la table de vérité définit une assignation pour les variables des formules présentes dans les colonnes de la table et chaque colonne donne la valeur d'une formule.

Exemple 1.2.4 *Nous donnons la table de vérité des formules suivantes : $x \Rightarrow y$, $\neg x$, $\neg x \vee y$, $(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$ et $x \vee \neg y$.*

x	y	$x \Rightarrow y$	$\neg x$	$\neg x \vee y$	$(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$	$x \vee \neg y$
0	0					
0	1					
1	0					
1	1					

1.2.3 Définitions et notions élémentaires de logique

Nous listons les notions élémentaires de la logique. Nous illustrons par des exemples et contre-exemples chacune des définitions introduites afin d'en faire comprendre les subtilités.

Définition 1.2.5 (Formules équivalentes) *Deux formules A et B sont* équivalentes *si elles ont la même valeur pour toute assignation.*

Exemple 1.2.6 *Les colonnes des deux formules $x \Rightarrow y$ et $\neg x \vee y$ sont identiques dans l'exemple 1.2.4 page précédente. Ces deux formules sont donc équivalentes. Par contre les formules $x \Rightarrow y$ et $x \vee \neg y$ ne sont pas équivalentes car elles n'ont pas les mêmes tables de vérité.*

Remarque 1.2.7 *Nous n'utilisons pas le symbole du connecteur logique $\Leftrightarrow$ pour dire que A et B sont équivalentes. Nous notons que les formules A et B sont équivalentes par $A \equiv B$ ou simplement $A = B$ si le contexte nous permet de comprendre que le signe égal indique l'équivalence. Ainsi, $x \Rightarrow y = \neg x \vee y$ signifie que la formule $x \Rightarrow y$ est équivalente à la formule $\neg x \vee y$.*

Définition 1.2.8 (Valide, tautologie) *Une formule est* valide *si elle a la valeur* 1 *pour toute assignation. Une formule valide est aussi appelée une* tautologie.

Exemple 1.2.9 *En regardant la table de vérité de l'exemple 1.2.4 page précédente nous obtenons que :*
- *la formule $(x \Rightarrow y) \Leftrightarrow (\neg x \vee y)$ est valide ;*
- *la formule $x \Rightarrow y$ n'est pas valide car*

Notation : $\models A$ dénote le fait que la formule A est valide. Nous pouvons écrire $\models x \vee \neg x$, car $x \vee \neg x$ est une tautologie.

Propriété 1.2.10 *Les formules A et B sont équivalentes si et seulement si la formule $A \Leftrightarrow B$ est valide.*

Preuve : La propriété est une conséquence de la table 1.1 page précédente et des définitions précédentes.

$\Rightarrow$ Si les formules A et B sont équivalentes cela signifie qu'elles ont la même table de vérité, ainsi d'après la définition du connecteur $\Leftrightarrow$ donnée dans la table 1.1 page précédente la table de vérité de $A \Leftrightarrow B$ contient uniquement des 1 donc $A \Leftrightarrow B$ est valide.

$\Leftarrow$ Si la formule $A \Leftrightarrow B$ est valide, nous déduisons que la table de vérité de $A \Leftrightarrow B$ contient uniquement des 1, ainsi d'après la définition du connecteur $\Leftrightarrow$ donnée dans la table 1.1 page précédente les tables de vérité de A et de B coïncident donc les formules A et B sont équivalentes.

□

Définition 1.2.11 (Modèle d'une formule) *Une assignation v qui donne la valeur* 1 *à une formule est un* modèle *de la formule. Nous dirons aussi que v* satisfait *la formule ou que v rend* vraie *la formule.*

Exemple 1.2.12 *$x \Rightarrow y$ a pour modèle*

Cette notion de modèle s'étend aux ensembles de formules comme suit.

Définition 1.2.13 (Modèle d'un ensemble de formules) *Une assignation est un* modèle d'un ensemble de formules *si et seulement si elle est un modèle de chaque formule de l'ensemble.*

Exemple 1.2.14

Propriété 1.2.15 *Une assignation est un modèle d'un ensemble de formules si et seulement si elle est un modèle de la conjonction des formules de l'ensemble.*

Preuve : La preuve est demandée dans l'exercice 11 page 33. □

Exemple 1.2.16 *L'ensemble de formules* $\{a \Rightarrow b, b \Rightarrow c\}$ *et la formule* $(a \Rightarrow b) \wedge (b \Rightarrow c)$ *ont les mêmes modèles.*

Définition 1.2.17 (Contre-Modèle) *Une assignation v qui donne la valeur* 0 *à une formule est un* contre-modèle *de la formule. Nous dirons que v ne satisfait pas la formule ou v rend la formule* fausse.

Exemple 1.2.18 $x \Rightarrow y$ *a pour contre-modèle*

Remarque 1.2.19 (Contre-modèle d'un ensemble de formules) *La notion de contre-modèle s'étend aux ensembles de formules de la même manière que la notion de modèle.*

Définition 1.2.20 (Formule satisfaisable) *Une formule (respectivement un ensemble de formules) est* satisfaisable *s'il existe une assignation qui en est un modèle.*

Définition 1.2.21 (Formule insatisfaisable) *Une formule (respectivement un ensemble de formules) est* insatisfaisable *si elle (respectivement s'il) n'est pas satisfaisable.*

Une formule (respectivement un ensemble de formules) insatisfaisable ne possède pas de modèle. Sa table de vérité ne comporte que des 0. La négation d'une tautologie est donc une formule insatisfaisable.

Exemple 1.2.22

Remarque 1.2.23 *Les logiciens utilisent le mot* consistant *comme synonyme de satisfaisable et* contradictoire *comme synonyme d'insatisfaisable.*

Définition 1.2.24 (Conséquence) *Soient* Γ *un ensemble de formules et A une formule : A est* conséquence *de l'ensemble* Γ *d'hypothèses si tout modèle de* Γ *est modèle de A. Le fait que A soit conséquence de* Γ *est noté par* $\Gamma \models A$.

Exemple 1.2.25 *D'après la table de vérité suivante, la formule* $a \Rightarrow c$ *est conséquence des hypothèses* $a \Rightarrow b$ *et* $b \Rightarrow c$.

a	b	c	$a \Rightarrow b$	$b \Rightarrow c$	$a \Rightarrow c$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Remarque 1.2.26 (Validité et conséquence) *Nous notons A est valide par* $\models A$, *car A est valide si et seulement si A est conséquence de l'ensemble vide.*

Maintenant, nous établissons l'équivalence de la validité d'une formule composée d'hypothèses et d'une conclusion avec la conséquence de la conclusion à partir des hypothèses mais aussi avec l'insatifaisabilité des hypothèses et de la négation de la conclusion. Ces relations sont constamment utilisées dans les exercices et les démonstrations.

Propriété 1.2.27 *Soient* $n+1$ *formules* $A_1, \ldots, A_n$, *B. Soit* H_n *la* conjonction *des formules* $A_1, \ldots, A_n$. *Les trois formulations suivantes sont équivalentes :*

1. $A_1, \ldots, A_n \models B$, *c'est-à-dire B est conséquence des hypothèses* $A_1, \ldots, A_n$.
2. *La formule* $H_n \Rightarrow B$ *est valide.*

3. *$H_n \wedge \neg B$ est insatisfaisable.*

Preuve : La propriété est une conséquence de la table 1.1 page 15 et des définitions précédentes. Soient $n+1$ formules $A_1, \ldots A_n, B$ et $H_n = A_1 \wedge \ldots \wedge A_n$.

1 ⇒ 2 : Supposons que $A_1, \ldots, A_n \models B$, c'est-à-dire tout modèle de $A_1, \ldots, A_n$ est aussi modèle de B.

Nous obtenons donc dans tous les cas que $H_n \Rightarrow B$ est valide.

2 ⇒ 3 : Supposons que $H_n \Rightarrow B$ est valide, cela signifie que pour toute assignation v, $[H_n \Rightarrow B]_v = 1$. D'après la table 1.1 page 15 nous avons deux possibilités soit $[H_n]_v = 0$, soit $[H_n]_v = 1$ et $[B]_v = 1$. Or $[H_n \wedge \neg B]_v = \min([H_n]_v, [\neg B]_v) = \min([H_n]_v, 1 - [B]_v)$. Dans les deux cas, nous obtenons $[H_n \wedge \neg B]_v = 0$. Nous concluons donc que $H_n \wedge \neg B$ est insatisfaisable.

3 ⇒ 1 : Supposons que $H_n \wedge \neg B$ est insatisfaisable, c'est-à-dire que pour toute assignation la formule $H_n \wedge \neg B$ est contradictoire. Montrons que les modèles de $A_1, \ldots, A_n$ sont aussi des modèles de B.

Soit v une assignation modèle de $A_1, \ldots, A_n$, il en découle que $[A_i]_v = 1$ pour $i = 1, \ldots, n$ donc $[H_n]_v = [A_1 \wedge \ldots \wedge A_n]_v = 1$. D'après notre hypothèse, nous en déduisons que $[\neg B]_v = 0$. D'où, $1 - [B]_v = 0$. Nous avons donc $[B]_v = 1$, c'est-à-dire que v est aussi un modèle de B.

En utilisant le résultat de l'exercice 7 page 32 nous concluons. □

Exemple 1.2.28 *Nous considérons les deux formules $a \Rightarrow b$ et $b \Rightarrow c$, nous illustrons le théorème précédent en prouvant que $a \Rightarrow b, b \Rightarrow c \models a \Rightarrow c$, soit en montrant que $(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$ est une tautologie, soit que $(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge \neg(a \Rightarrow c)$ est insatisfaisable. Pour cela nous donnons la table de vérité associée à ces formules.*

a	b	c	$a \Rightarrow b$	$b \Rightarrow c$	$a \Rightarrow c$	$(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$	$(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge \neg(a \Rightarrow c)$
0	0	0	1	1	1		
0	0	1	1	1	1		
0	1	0	1	0	1		
0	1	1	1	1	1		
1	0	0	0	1	0		
1	0	1	0	1	1		
1	1	0	1	0	0		
1	1	1	1	1	1		

1.2.5 Équivalences remarquables

Raisonner par équivalence c'est utiliser les propriétés de l'équivalence (réflexivité, symétrie, transitivité), et la propriété de remplacement d'une formule par une autre formule équivalente pour obtenir de nouvelles équivalences à partir des équivalences déjà prouvées ou admises. Ci-dessous, nous listons des équivalences remarquables de la logique.

1. La disjonction est :
   — Associative, c'est-à-dire, $x \vee (y \vee z) \equiv (x \vee y) \vee z$.
   — Commutative, c'est-à-dire, $x \vee y \equiv y \vee x$.
   — 0 est l'élément neutre de la disjonction, c'est-à-dire, $0 \vee x \equiv x$.
2. La conjonction est :
   — Associative, c'est-à-dire, $x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$.

— Commutative, c'est-à-dire, $x \wedge y \equiv y \wedge x$.
— 1 est l'élément neutre de la conjonction, c'est-à-dire, $1 \wedge x \equiv x$.

3. La conjonction est distributive sur la disjonction : $x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$.
4. La disjonction est distributive sur la conjonction : $x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$.
5. Les lois de la négation :
   — $x \wedge \neg x \equiv 0$.
   — $x \vee \neg x \equiv 1$ (Le *tiers-exclus*).
6. $\neg\neg x \equiv x$.
7. $\neg 0 \equiv 1$.
8. $\neg 1 \equiv 0$.
9. Les lois de De Morgan :
   — $\neg(x \wedge y) \equiv \neg x \vee \neg y$.
   — $\neg(x \vee y) \equiv \neg x \wedge \neg y$.
10. 0 est l'élément absorbant de la conjonction : $0 \wedge x \equiv 0$.
11. 1 est l'élément absorbant de la disjonction : $1 \vee x \equiv 1$.
12. idempotence de la disjonction : $x \vee x \equiv x$.
13. idempotence de la conjonction : $x \wedge x \equiv x$.

Les équivalences 1 page ci-contre à 5 se démontrent à l'aide de tables de vérité[3]. Les équivalences 6 à 13 se déduisent des équivalences 1 page ci-contre à 5, comme nous le montrons dans la section 1.5 page 24. Ci-dessous, nous donnons quelques lois de simplification qui permettent d'alléger les raisonnements logiques.

Propriété 1.2.31 (Lois de simplification) *Pour tout x, y nous avons :*
— $x \vee (x \wedge y) \equiv x$.
— $x \wedge (x \vee y) \equiv x$.
— $x \vee (\neg x \wedge y) \equiv x \vee y$.
— $x \wedge (\neg x \vee y) \equiv x \wedge y$.

Preuve : En utilisant les équivalences remarquables nous prouvons les trois lois de simplification. La preuve est demandée dans l'exercice 12 page 33. □

1.3 Substitution et remplacement

Dans ce paragraphe, nous étendons l'ensemble des formules valides par substitution et remplacement.

1.3.1 Substitution

Définition 1.3.1 (Substitution) *Une* substitution *est une application de l'ensemble des variables dans l'ensemble des formules. L'application d'une substitution σ à une formule consiste à remplacer dans la formule toute variable x par la formule $\sigma(x)$.*

Notation : Soit A une formule, nous notons $A\sigma$ ou $\sigma(A)$ l'application de la substitution σ à la formule A.

Définition 1.3.2 (Support d'une substitution, substitution à support fini) *Le* support *d'une substitution σ est l'ensemble des variables x telles que $x\sigma \neq x$. Une substitution σ à* support fini *est notée $< x_1 := A_1, \ldots, x_n := A_n >$, où $A_1, \ldots, A_n$ sont des formules, $x_1, \ldots, x_n$ sont des variables* distinctes *et la substitution vérifie :*
— *pour i de 1 à n, $x_i\sigma = A_i$*
— *pour toute variable y telle que $y \notin \{x_1, \ldots, x_n\}$, nous avons : $y\sigma = y$*

3. Nous laissons le soin au lecteur de se convaincre de la véracité de ces équivalences en construisant les tables correspondantes.

Exemple 1.3.3 *Soient la formule* $A = x \vee x \wedge y \Rightarrow z \wedge y$ *et la substitution* $\sigma =< x := a \vee b, z := b \wedge c >$, σ *appliquée à A donne :*

Le support de σ *est fini et comporte les variables x et z.*

Propriété 1.3.4 *Soient A une formule, v une assignation et* σ *une substitution, nous avons* $[A\sigma]_v = [A]_w$ *où pour toute variable x,* $w(x) = [\sigma(x)]_v$.

Preuve : Soient A une formule, v une assignation et σ une substitution, nous voulons montrer que $[A\sigma]_v = [A]_w$ où pour toute variable x, $w(x) = [\sigma(x)]_v$. Par une récurrence sur la taille des formules, nous obtenons :

Cas de base : $|A| = 0$.

Induction :

Supposons que la propriété soit vraie pour toute formule de taille inférieure ou égale à n, montrons qu'elle est vraie pour une formule A de taille $n+1$:

□

Exemple 1.3.5 *Soit* $A = x \vee y \vee d$ *une formule. Soit* $\sigma =< x := a \vee b, y := b \wedge c >$ *une substitution. Soit v une assignation telle que* $v(a) = 1$, $v(b) = 0$, $v(c) = 0$, $v(d) = 0$. *Nous avons :*

Théorème 1.3.6 *L'application d'une substitution à une formule valide donne une formule valide.*

Preuve : Soient A une formule valide, σ une substitution et v une assignation quelconque. D'après la propriété 1.3.4 : $[A\sigma]_v = [A]_w$ où pour toute variable x, $w(x) = [\sigma(x)]_v$. Puisque A est valide, $[A]_w = 1$. Par suite $A\sigma$ vaut 1 dans toute assignation, c'est donc une formule valide. □

Exemple 1.3.7 *Soit A la formule* $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$. *Cette formule est valide, c'est une équivalence remarquable. Soit* σ *la substitution suivante :* $< p := (a \vee b), q := (c \wedge d) >$. *La formule*

La substitution est définie sur les formules, pour l'appliquer sans erreur aux formules à priorité, il suffit que l'image, par la substitution, de toute variable soit une variable, une constante ou une formule entre parenthèses.

1.3.2 Remplacement

La notion de substitution ne nous permet pas de remplacer une formule par une formule, nous introduisons la notion de remplacement à cette fin.

Définition 1.3.8 (Remplacement) *Soient A, B, C, D des formules. La formule D est obtenue en remplaçant dans C certaines occurrences de A par B, s'il existe une formule E et une variable x telles que, $C = E < x := A >$ et $D = E < x := B >$.*

Exemple 1.3.9 *Considérons la formule $C = ((a \Rightarrow b) \vee \neg(a \Rightarrow b))$.*

— *La formule obtenue en remplaçant toutes les occurrences de $(a \Rightarrow b)$ par $(a \wedge b)$ dans C est*

$\boxed{}$

elle est obtenue en considérant la formule $E = (x \vee \neg x)$ et les substitutions $< x := (a \wedge b) >$ et $< x := (a \Rightarrow b) >$.

— *La formule obtenue en remplaçant la première occurrence de $(a \Rightarrow b)$ par $(a \wedge b)$ dans C est*

$\boxed{}$

elle est obtenue en considérant la formule $E = (x \vee \neg(a \Rightarrow b))$ et les substitutions $< x := (a \wedge b) >$ et $< x := (a \Rightarrow b) >$.

La différence entre une substitution et un remplacement est qu'une substitution remplace un ensemble de variables par des formules alors qu'un remplacement remplace les occurrences de certaines formules par une autre formule en utilisant des substitutions.

Théorème 1.3.10 *Soient C une formule et D la formule obtenue en remplaçant, dans C, des* occurrences *de la formule A par la formule B, alors $(A \Leftrightarrow B) \models (C \Leftrightarrow D)$.*

Preuve : Par définition du remplacement, il existe une formule E et une variable x telles que, $C = E < x := A >$ et $D = E < x := B >$. Supposons que v est une assignation modèle de $(A \Leftrightarrow B)$. Nous avons donc $[A]_v = [B]_v$. D'après la propriété 1.3.4 page ci-contre :

— $[C]_v = [E]_w$ où w est identique à v sauf que $w(x) = [A]_v$

— $[D]_v = [E]_{w'}$ où w' est identique à v sauf que $w'(x) = [B]_v$

Puisque $[A]_v = [B]_v$, les assignations w et w' sont identiques, donc $[C]_v = [D]_v$. Par suite v est modèle de $(C \Leftrightarrow D)$. □

Corollaire 1.3.11 *Soient C une formule et D la formule obtenue en remplaçant, dans C, une* occurrence *de la formule A par la formule B, alors $A \equiv B$ implique $C \equiv D$.*

Preuve : Si $A \equiv B$, alors la formule $(A \Leftrightarrow B)$ est valide (propriété 1.2.10), donc la formule $(C \Leftrightarrow D)$ également puisqu'elle est, d'après le théorème ci-dessus, la conséquence de $(A \Leftrightarrow B)$, par suite $C \equiv D$. □

Exemple 1.3.12 *Le remplacement d'une occurrence d'une formule A par une occurrence de B est mis en évidence par des boîtes marquant ces occurrences.*

— *D'après théorème 1.3.10 : $p \Leftrightarrow q \models (p \vee (\boxed{p} \Rightarrow r)) \Leftrightarrow (p \vee (\boxed{q} \Rightarrow r))$.*

— *D'après le corollaire 1.3.11 : $(\neg(p \vee q) \Rightarrow (\boxed{\neg(p \vee q)} \vee r)) \equiv (\neg(p \vee q) \Rightarrow (\boxed{(\neg p \wedge \neg q)} \vee r))$, puisque $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$.*

Remarque 1.3.13 *Le théorème précédent et son corollaire s'appliquent aux formules. Quand nous faisons un remplacement directement sur une formule à priorité, nous devons nous assurer que ce remplacement reste correct sur les formules (strictes) sous peine de commettre des erreurs. Par exemple, considérons les deux équivalences $a \wedge \boxed{b} \equiv a \wedge \boxed{b}$ et $\neg c \Rightarrow d \equiv c \vee d$. Remplaçons b à gauche par $\neg c \Rightarrow d$ et à droite par $c \vee d$. Nous observons que bien que $\neg c \Rightarrow d \equiv c \vee d$, $a \wedge \neg c \Rightarrow d \not\equiv a \wedge c \vee d$, car pour $a = c = d = 0$, la formule de gauche vaut 1 et celle de droite vaut 0. Ici le corollaire ne doit pas être appliqué à l'occurrence encadrée car $a \wedge \neg c \Rightarrow d$ est une abréviation de $((a \wedge \neg c) \Rightarrow d)$, donc $\neg c \Rightarrow d$ n'apparaît pas comme une occurrence possible de $a \wedge \neg c \Rightarrow d$.*

Nous avons défini les remplacements à partir des substitutions, nous allons maintenant appliquer les remplacements à une formule afin de la transformer en une formule en forme normale équivalente.

1.4 Formes Normales

Mettre une formule en forme normale consiste à la transformer en une formule équivalente ayant des propriétés structurelles. Nous introduisons deux notions de formes normales : la forme normale disjonctive qui permet de mettre en évidence les modèles et la forme normale conjonctive qui exhibe les contre-modèles. La définition de forme normale nécessite l'introduction des concepts de littéral, monôme et clause.

Définition 1.4.1 (Littéral, monôme, clause)
— *Un* littéral *est une variable ou la négation d'une variable.*
— *Un* monôme *est une conjonction de littéraux.*
— *Une* clause *est une disjonction de littéraux.*

Exemple 1.4.2 *Nous illustrons par quelques exemples simples ces nouvelles notions :*
— $x, y, \neg z$ *sont des littéraux.*
— $x \wedge \neg y \wedge z$ *est un monôme dont l'unique modèle est* $x = 1, y = 0, z = 1$.
— *Le monôme* $x \wedge \neg y \wedge z \wedge \neg x$ *comporte une variable et sa négation : il vaut* 0.
— $x \vee \neg y \vee z$ *est une clause dont l'unique contre modèle est* $x = 0, y = 1, z = 0$.
— *La clause* $x \vee \neg y \vee z \vee \neg x$ *comporte une variable et sa négation : elle vaut* 1.

1.4.1 Transformation en forme normale

Nous introduisons la notion de forme normale et montrons qu'il est toujours possible de transformer une formule en une formule équivalente en forme normale.

Définition 1.4.3 (Forme normale) *Une formule est en* forme normale *si elle n'utilise que les opérateurs* $\wedge, \vee, \neg$ *et que les négations sont uniquement appliquées aux variables.*

Exemple 1.4.4 *La formule* $\neg a \vee b$ *est en forme normale, alors que la formule* $a \Rightarrow b$ *n'est pas en forme normale bien qu'elle soit équivalente à la première.*

Nous expliquons maintenant comment transformer toute formule en une formule en forme normale équivalente grâce à des remplacements. Appliquer l'équivalence $A \equiv B$ à la formule C, c'est remplacer dans C une occurrence de A par une occurrence de B : nous avons prouvé dans le théorème 1.3.10 page précédente qu'un tel remplacement change C en une formule équivalente. Ainsi pour transformer une formule en une formule équivalente de forme normale, nous appliquons les transformations suivantes :

1. **Élimination des équivalences** : remplacer une occurrence de $A \Leftrightarrow B$ par l'une des sous-formules :
   (a) $(\neg A \vee B) \wedge (\neg B \vee A)$.
   (b) $(A \wedge B) \vee (\neg A \wedge \neg B)$.
2. **Élimination des implications** : remplacer une occurrence de $A \Rightarrow B$ par $\neg A \vee B$.
3. **Déplacement des négations** : remplacer une occurrence de
   (a) $\neg\neg A$ par A.
   (b) $\neg(A \vee B)$ par $\neg A \wedge \neg B$.
   (c) $\neg(A \wedge B)$ par $\neg A \vee \neg B$.
   ainsi les négations ne portent que sur des variables.

En appliquant ces trois transformations dans l'ordre indiqué, il est clair que la formule initiale a été transformée en une formule en forme normale équivalente. Dans l'exercice 25 page 34, nous prouvons que l'ordre des transformations n'est pas important : en effectuant les transformations ci-dessus dans un ordre quelconque, nous obtenons finalement une formule équivalente en forme normale.

Remarque 1.4.5 *Il est par exemple recommandé de remplacer une sous-formule de la forme* $\neg(A \Rightarrow B)$ *par* $A \wedge \neg B$*, ce qui en fait combine une élimination de l'implication et un déplacement de la négation. En pratique, il est plus efficace de simplifier le plus tôt possible de la façon suivante :*
— *Remplacer par* 0 *une conjonction qui comporte soit comporte une formule et sa négation, soit un* 0.
— *Remplacer par* 1 *une disjonction qui comporte soit une formule et sa négation, soit un* 1.
— *Remplacer* $\neg 1$ *par* 0 *et* $\neg 0$ *par* 1.
— *Enlever les* 0 *des disjonctions et les* 1 *des conjonctions.*
— *Appliquer les simplifications* $x \vee (x \wedge y) \equiv x$, $x \wedge (x \vee y) \equiv x$, $x \vee (\neg x \wedge y) \equiv x \vee y$.
— *Appliquer l'idempotence de la disjonction et de la conjonction.*

1.4.2 Transformation en forme normale disjonctive (somme de monômes)

La forme normale disjonctive permet de trouver facilement des modèles.

Définition 1.4.6 (Forme normale disjonctive) *Une formule est une* forme normale disjonctive *(en bref fnd) si et seulement si elle est une disjonction (somme) de monômes.*

L'intérêt des formes normales disjonctives est de mettre en évidence les modèles.

Exemple 1.4.7 $(x \wedge y) \vee (\neg x \wedge \neg y \wedge z)$ *est une fnd composée de deux monômes, chacun d'eux nous donne un des deux modèles possibles :*

En partant d'une forme normale et en distribuant toutes les conjonctions sur les disjonctions, nous obtenons une disjonction de monômes. Pour cela il faut aussi savoir regrouper plusieurs applications de la distributivité.

Exemple 1.4.8 *Dans une transformation en disjonction de monômes, nous pouvons appliquer de gauche à droite l'équivalence* $(a \vee b) \wedge (c \vee d \vee e) \equiv$

La transformation par équivalence d'une formule en une disjonction de monômes peut être utilisée pour déterminer si une formule est valide ou non. Soit A une formule dont nous souhaitons déterminer la validité : Nous transformons $\neg A$ en une disjonction de monômes *équivalente* à $\neg A$. Si $\neg A = 0$ alors $A = 1$ donc A est valide, sinon, toutes simplifications étant faites, $\neg A$ est égal à une disjonction de monômes non nuls, qui nous donnent des modèles de $\neg A$, donc des contre-modèles de A.

Exemple 1.4.9 *Soit* $A = (p \Rightarrow (q \Rightarrow r)) \Rightarrow (p \wedge q \Rightarrow r)$. *Déterminons si A est valide par transformation de* $\neg A$ *en disjonction de monômes.*

Exemple 1.4.10 *Soit* $A = ((a \Rightarrow b) \wedge c) \vee (a \wedge d)$. *Déterminer si A est valide par transformation de* $\neg A$ *en disjonction de monômes.*

$\neg A$	
$\equiv \neg((a \Rightarrow b) \wedge c) \wedge \neg(a \wedge d)$	*par déplacement des négations*
$\equiv (\neg(a \Rightarrow b) \vee \neg c) \wedge (\neg a \vee \neg d)$	*par déplacement des négations*
$\equiv (a \wedge \neg b \vee \neg c) \wedge (\neg a \vee \neg d)$	*par combinaison du déplacement d'une négation et élimination de l'implication*
$\equiv (a \wedge \neg b \wedge \neg a) \vee (a \wedge \neg b \wedge \neg d) \vee (\neg c \wedge \neg a) \vee (\neg c \wedge \neg d)$	*par distributivité de la disjonction sur la conjonction*
$\equiv (a \wedge \neg b \wedge \neg d) \vee (\neg c \wedge \neg a) \vee (\neg c \wedge \neg d)$	*par simplification*

Nous obtenons 3 *modèles de* $\neg A$ *(*$a = 1, b = 0, d = 0$*;* $a = 0, c = 0$*;* $c = 0, d = 0$*), c'est-à-dire, des contre-modèles de A. Donc A n'est pas valide.*

1.4.3 Transformation en forme normale conjonctive (produit de clauses)

La forme normale conjonctive permet d'exhiber facilement des contre-modèles.

Définition 1.4.11 (Forme normale conjonctive) *Une formule est une* forme normale conjonctive *(en bref fnc) si et seulement si elle est une conjonction (produit) de clauses.*

L'intérêt des formes normales conjonctives est de mettre en évidence des contre-modèles.

Exemple 1.4.12 $(x \vee y) \wedge (\neg x \vee \neg y \vee z)$ *est une fnc, qui a deux contre-modèles*

Par convention, nous considérons que 0 et 1 sont des disjonctions de monômes et des conjonctions de clauses. Nous appliquons la distributivité (inhabituelle) de la disjonction sur la conjonction, autrement dit nous remplaçons toute sous-formule $A \vee (B \wedge C)$ par $(A \vee B) \wedge (A \vee C)$, et toute sous-formule $(B \wedge C) \vee A$ par $(B \vee A) \wedge (C \vee A)$.

Exemple 1.4.13 *Nous reprenons l'exemple* *1.4.8 page précédente* *et obtenons* $(a \wedge b) \vee (c \wedge d \wedge e) \equiv$

La transformation par équivalence d'une formule en une conjonction de clauses de littéraux peut aussi être utilisée pour déterminer si une formule est valide ou non. Soit A une formule dont nous souhaitons déterminer la validité : Nous transformons A en une conjonction de clauses *équivalente* à A. Si $A = 1$ alors A est valide, sinon, toutes simplifications étant faites, A est égale à une conjonction de clauses non égales à 1, et chacune de ces disjonctions nous donne un contre-modèle de A.

1.5 Algèbre de Boole

Cette notion fut introduite par le mathématicien britannique George Boole au milieu du XIXe siècle. Elle permet notamment de traduire les propositions en équations (généralement, cette écriture est plus concise). Nous rappelons d'abord la définition d'une algèbre de Boole. Nous déduisons alors que la logique propositionnelle est une algèbre de Boole. Ensuite nous prouvons certaines propriétés usuelles des algèbres de Boole. Nous terminons cette section en présentant la notion de dualité.

1.5.1 Définition et notations

La définition d'algèbre de Boole donnée ci-dessous est minimale en nombre d'axiomes.

Définition 1.5.1 *Une* algèbre de Boole *est un ensemble d'au moins deux éléments,* 0, 1, *et trois opérations, complément (le complément de x est noté* $\overline{x}$*), somme (+) et produit (.), qui vérifient les axiomes suivants :*

1. *la somme est :*
   — *associative :* $x + (y + z) = (x + y) + z$,
   — *commutative :* $x + y = y + x$,
   — 0 *est élément neutre de la somme :* $0 + x = x$,
2. *le produit est :*
   — *associatif :* $x.(y.z) = (x.y).z$,
   — *commutatif :* $x.y = y.x$,
   — 1 *est élément neutre du produit :* $1.x = x$,
3. *le produit est distributif sur la somme :* $x.(y + z) = (x.y) + (x.z)$,
4. *la somme est distributive sur le produit :* $x + (y.z) = (x + y).(x + z)$,
5. *les lois de la négation :*
   — $x + \overline{x} = 1$,
   — $x.\overline{x} = 0$.

Attention, le fait que la distributivité de la somme sur le produit n'est pas usuelle rend son application propice aux erreurs.

Comme indiqué précédemment dans la sous-section 1.2.5 page 18, nous pouvons prouver en utilisant des tables de vérité que la logique propositionnelle vérifie l'ensemble des axiomes d'une algèbre de Boole. Ainsi nous pouvons considérer la logique propositionnelle comme la plus petite algèbre de Boole, car elle contient deux éléments. De ce fait, nous pouvons utiliser les notations booléennes (plus condensées) en lieu et place des notations propositionnelles, comme indiqué dans la table de correspondance donnée dans la figure 1.1. Nous conseillons de les utiliser pour effectuer de gros calculs. Par ailleurs, observez que ces notations sont fréquemment utilisées dans le domaine du matériel.

notations logiques	notations booléennes
$\neg a$	$\overline{a}$
$a \wedge b$	$a.b$
$a \vee b$	$a+b$
$a \Rightarrow b$	$\overline{a}+b$
$a \Leftrightarrow b$	$a.b+(\overline{a}.\overline{b})$ ou $(a+\overline{b}).(\overline{a}+b)$

FIGURE 1.1 – Correspondance entre l'algèbre de Boole et la logique propositionnelle.

Remarque 1.5.2 (Conventions d'écriture)

— *Parfois, la négation est utilisée à la place du complémentaire. Par exemple, $-a$ dénote $\overline{a}$ et $-(a+b)$ représente $\overline{a+b}$.*

— *Afin de permettre l'omission du point, dans les notations booléennes, les variables sont souvent dénotées par une seule lettre (avec ou sans indice). Ainsi lorsque a et b sont des variables, des formules entre parenthèses ou sous une barre de négation, nous abrégeons a.b en ab suivant une pratique usuelle en mathématiques.*

— *Enfin l'équivalence de deux formules est systématiquement notée avec le symbole $=$ au lieu du symbole $\equiv$.*

Il est important de noter que la logique propositionnelle n'est pas l'unique algèbre de Boole. Par exemple, l'ensemble $\mathcal{P}(X)$ des sous-ensembles d'un ensemble X est une algèbre de Boole, comme nous l'indique la mise en correspondance des opérateurs ensemblistes avec leur désignation dans l'algèbre de Boole dans la figure 1.2.

Algèbre de Boole	$\mathcal{P}(X)$
1	X
0	$\emptyset$
$\overline{p}$	$X-p$
$p+q$	$p \cup q$
$p.q$	$p \cap q$

FIGURE 1.2 – Correspondance entre l'algèbre de Boole et $\mathcal{P}(X)$.

Puisque les lois de simplification sont déduites des lois de l'algèbre de Boole, comme nous allons le montrer dans la suite, elles sont vérifiées dans toute algèbre de Boole. En particulier dans l'algèbre de Boole $\mathcal{P}(X)$ nous avons $A \cup (A \cap B) = A$. Une *preuve dans l'algèbre de Boole* est une suite d'égalités, permettant de passer d'une formule à une autre formule équivalente en utilisant les axiomes ou simplifications de l'algèbre de Boole.

1.5.2 Propriétés

Nous présentons certaines des propriétés qui découlent de la définition de l'algèbre de Boole et qui sont couramment utilisées dans les raisonnements. Pour raccourcir les preuves de ces propriétés, nous utilisons implicitement l'associativité et la commutativité de la somme et du produit.

Propriété 1.5.3 *Dans une algèbre de Boole, pour tout x, il y a un et un seul y tel que $x+y=1$ et $xy=0$, autrement dit la négation est unique.*

Preuve : Il y a un élément y vérifiant $x+y=1$ et $xy=0$, c'est $\overline{x}$ d'après les axiomes de l'algèbre de Boole. Montrons qu'il est unique. Raisonnons par contradiction, supposons qu'il n'est pas unique, nous avons donc qu'il existe y et z deux éléments distincts tels que $x+y=1$, $xy=0$, $x+z=1$, et $xz=0$. Nous considérons $u=xy+z$. Puisque $xy=0$, nous avons : $u=z$. Par distributivité de la somme sur le produit, nous avons : $u=(x+z)(y+z)$. Puisque $x+z=1$ et que 1 est l'élément neutre du produit, nous avons : $u=y+z$. Nous concluons donc que $z=y+z$. En considérant $v=xz+y$ et échangeant les rôles de y et z, nous obtenons de façon analogue $y=y+z$. Par suite $y=z$, ce qui prouve donc l'unicité de la négation. □

Propriété 1.5.4 *Dans une algèbre de Boole, les équivalences suivantes sont vraies pour tout x et tout y :*

1. $\overline{1}=0$.
2. $\overline{0}=1$.
3. $\overline{\overline{x}}=x$.
4. *Idempotence du produit :* $x.x=x$.
5. *Idempotence de la somme :* $x+x=x$.
6. 1 *est élément absorbant de la somme :* $1+x=1$.
7. 0 *est élément absorbant du produit :* $0.x=0$.
8. *Lois de De Morgan :*
   — $\overline{xy}=\overline{x}+\overline{y}$.
   — $\overline{x+y}=\overline{x}.\overline{y}$.

Preuve :

1. $\overline{1}=0$.
2. $\overline{0}=1$.
3. $\overline{\overline{x}}=x$.
4. Idempotence du produit : $x.x=x$.

5. Idempotence de la somme : $x+x=x$.

6. 1 est élément absorbant de la somme : $1+x=1$.

7. 0 est élément absorbant du produit : $0.x=0$.

8. Lois de De Morgan :

   — $\overline{x.y}=\overline{x}+\overline{y}$:

— $\overline{x+y} = \overline{x}.\overline{y}$.

□

1.6 Fonctions booléennes

Nous regardons maintenant nos formules comme des fonctions mathématiques sur le domaine $\mathbb{B} = \{0,1\}$. Nous définissons formellement ce qu'est une fonction booléenne, puis nous revisitons les formes normales conjonctives et disjonctives avec ce formalisme. Remarquons que les fonctions booléennes sont des notions utilisées en cryptologie par exemple dans la construction de chiffrements symétriques (le chiffrement One-Time-Pad, un des chiffrements les plus sûrs est basé sur la fonction booléenne du « ou-exclusif »).

Définition 1.6.1 (Fonction booléenne) *Une* fonction booléenne *f est une fonction dont les arguments et le résultat sont dans le domaine $\mathbb{B} = \{0,1\}$.*

$$f : \mathbb{B}^n \to \mathbb{B}$$

Exemple 1.6.2 *Nous donnons des exemples et contre-exemples de fonctions booléennes :*
- *La fonction $f : \mathbb{B} \to \mathbb{B} : f(x) = \neg x$ est une fonction booléenne.*
- *La fonction $f : \mathbb{N} \to \mathbb{B} : f(x) = x \bmod 2$ n'est pas une fonction booléenne.*
- *La fonction $f : \mathbb{B} \to \mathbb{N} : f(x) = x+1$ n'est pas une fonction booléenne.*
- *La fonction $f : \mathbb{B} \times \mathbb{B} \to \mathbb{B} : f(x,y) = \neg(x \wedge y)$ est une fonction booléenne.*

1.6.1 Fonctions booléennes et somme de monômes

Grâce aux fonctions booléennes, nous sommes capables à partir des valeurs de vérité d'une formule de reconstruire la somme de monômes correspondante. Pour toute variable x, nous posons $x^0 = \overline{x}$ et $x^1 = x$.

Théorème 1.6.3 *Soit f une fonction booléenne à n arguments. Cette fonction est représentée à l'aide de n variables $x_1, \ldots, x_n$. Soit A la formule suivante :*

$$A = \sum_{f(a_1,\ldots,a_n)=1} x_1^{a_1} \ldots x_n^{a_n}.$$

Les a_i sont des valeurs booléennes et A est la somme des monômes $x_1^{a_1} \ldots x_n^{a_n}$ tels que $f(a_1,\ldots,a_n) = 1$. Par convention, si la fonction f vaut toujours 0 alors $A = 0$.

Pour toute assignation v telle que $v(x_1) = a_1, \ldots, v(x_n) = a_n$, nous avons $f(a_1,\ldots,a_n) = [A]_v$. En omettant la distinction entre une variable et sa valeur, ce résultat s'écrit plus brièvement : $f(x_1,\ldots x_n) = A$.

Exemple 1.6.4 *La fonction maj à 3 arguments vaut 1 lorsqu'au moins 2 de ses arguments valent 1. Nous donnons la table représentant cette fonction :*

x_1	x_2	x_3	$maj(x_1,x_2,x_3)$					
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	**1**	**1**	0	0	0	**1**
1	0	0	0	0	0	0	0	0
1	0	1	**1**	0	**1**	0	0	**1**
1	1	0	**1**	0	0	**1**	0	**1**
1	1	1	**1**	0	0	0	**1**	**1**

Nous notons sur la table de vérité, que chaque monôme ne vaut 1 que sur une ligne de la table et que la somme des monômes est équivalente à $maj(x_1,x_2,x_3)$. La fonction maj peut donc s'exprimer par

1.6.2 Fonctions booléennes et produit de clauses

Nous effectuons le même raisonnement pour la forme duale.

Théorème 1.6.5 *Soit f une fonction booléenne à n arguments. Cette fonction est représentée à l'aide de n variables $x_1,\ldots,x_n$. Soit A la formule suivante :*

$$A = \prod_{f(a_1,\ldots,a_n)=0} x_1^{\overline{a_1}} + \ldots + x_n^{\overline{a_n}}.$$

Les a_i sont des valeurs booléennes et A est le produit des clauses $x_1^{\overline{a_1}} + \ldots + x_n^{\overline{a_n}}$ telles que $f(a_1,\ldots,a_n) = 0$. Par convention si la fonction f vaut toujours 1 alors $A = 1$.

Pour toute assignation v telle que $v(x_1) = a_1,\ldots,v(x_n) = a_n$, nous avons $f(a_1,\ldots,a_n) = [A]_v$. En omettant la distinction entre une variable et sa valeur, ce résultat s'écrit plus brièvement : $f(x_1,\ldots,x_n) = A$.

Exemple 1.6.6 *Nous représentons la fonction maj définie dans l'exemple 1.6.4 page précédente par un produit de clauses :*

x_1	x_2	x_3	$maj(x_1,x_2,x_3)$					
0	0	0	**0**	**0**	1	1	1	**0**
0	0	1	**0**	1	**0**	1	1	**0**
0	1	0	**0**	1	1	**0**	1	**0**
0	1	1	1	1	1	1	1	1
1	0	0	**0**	1	1	1	**0**	**0**
1	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

La fonction maj peut donc s'exprimer en produit de clauses comme suit :

1.7 L'outil BDDC

`BDDC` (*Binary Decision Diagram based Calculator*) est un outil pour la manipulation de formules propositionnelles développé par Pascal Raymond et disponible à l'adresse suivante :

`http://www-verimag.imag.fr/~raymond/home/tools/bddc/`.

Cet outil est une « calculette propositionnelle » basée sur les *diagrammes de décision binaires*. Ces diagrammes permettent d'avoir une représentation interne des formules logiques sous la forme d'un graphe orienté acyclique ayant pour propriété d'être canonique. Cette calculette offre notamment la possibilité :

— d'évaluer si une formule est une tautologie,
— d'évaluer si une formule a un modèle,
— de transformer une formule en forme normale conjonctive,
— de transformer une formule en forme normale disjonctive,
— ...

Ainsi, il est possible d'utiliser `BDDC` pour résoudre plusieurs des exercices proposés ci-après, parmi lesquels les exercices 8 page 32 à 16 page 33 et 23 page 34 à 24 page 34.

1.8 Exercices

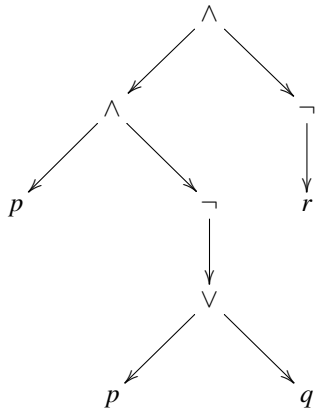
Exercice 1 (Formules strictes, formules à priorité) *Parmi les expressions suivantes, indiquer celles qui sont et celles qui ne sont pas des formules, au sens de la syntaxe stricte décrite dans la définition 1.1.1 page 12 :*

1. x
2. $(x$
3. (x)
4. $(x \Rightarrow (y \wedge z)$
5. $(x \Rightarrow (y \wedge z))$
6. $\neg(x \vee y)$
7. $(\neg(x \vee y))$
8. $\neg(x \Rightarrow y) \vee \neg(y \wedge z)$
9. $\neg(x)$
10. $(\neg((x \vee y) \wedge z) \Leftrightarrow (u \Rightarrow v)))$

□

Exercice 2 (Formules et priorités) *Montrer que toute formule stricte est une formule à priorité.* □

Exercice 3 (Formules et priorités) *L'arbre suivant représente la structure de la formule* $((p \wedge \neg(p \vee q)) \wedge \neg r)$.



En utilisant les priorités, elle peut s'écrire avec le moins de parenthèses possibles sous l'une des formes : $p \wedge \neg(p \vee q) \wedge \neg r$ *avec les notations logiques,* $p.\overline{p+q}.\overline{r}$ *avec les notations booléennes. Comme sur cet exemple, pour chacune des formules suivantes, indiquer sa structure sous forme d'arbre, sa forme la moins parenthésée avec les opérations logiques, sa forme la moins parenthésée avec les opérations booléennes :*

1. $(\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b))$.
2. $((\neg a \vee b) \wedge (\neg b \vee a))$.
3. $(((a \wedge b) \wedge c) \vee ((\neg a \wedge \neg b) \wedge \neg c))$.
4. $(p \Rightarrow (q \Rightarrow r))$.
5. $((p \Rightarrow q) \Rightarrow r)$.

□

Exercice 4 (Formules et priorités) *Indiquer sous forme d'arbres, les structures des formules à priorité suivantes :*

1. $p \Leftrightarrow \neg q \vee r$.
2. $p \vee q \Rightarrow r \wedge s$.
3. $p \vee q \Rightarrow r \Leftrightarrow s$.
4. $p \vee q \wedge r \Rightarrow \neg s$.
5. $p \Rightarrow r \wedge s \Rightarrow t$.

6. $p \vee q \wedge s \vee t$.
7. $p \wedge q \Leftrightarrow \neg r \vee s$.
8. $\neg p \wedge q \vee r \Rightarrow s \Leftrightarrow t$.

□

Exercice 5 (Hauteur,*) *Nous définissons la hauteur h d'une formule à priorité par :*

— $h(\top) = 0$ *et* $h(\bot) = 0$.
— *Si A est une variable* $h(A) = 0$.
— $h(\neg A) = 1 + h(A)$.
— $h((A)) = h(A)$.
— $h(A \circ B) = \max(h(A), h(B)) + 1$, *si* $\circ$ *est une des opérations* $\vee, \wedge, \Rightarrow, \Leftrightarrow$.

1. *Montrer que cette définition est ambigüe, c'est-à-dire trouver une formule qui peut avoir deux hauteurs différentes.*
2. *Proposer une nouvelle définition récursive de la hauteur n'ayant pas ce problème.*

□

Exercice 6 (Validité) *Donner les tables de vérités des formules suivantes :*

1. $p \Rightarrow (q \Rightarrow p)$.
2. $p \Rightarrow (q \Rightarrow r)$.
3. $(p \Rightarrow q) \Rightarrow (p \Rightarrow r)$.
4. $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$.

Indiquer celles qui sont valides. Indiquer les équivalences. □

Exercice 7 (Raisonnement circulaire) *Donner les tables de vérité des formules suivantes :* $a \Rightarrow b$, $b \Rightarrow c$, $c \Rightarrow a$, $a \Leftrightarrow b$, $b \Leftrightarrow c$ *et* $c \Leftrightarrow a$. *Conclure que* $(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge (c \Rightarrow a) \equiv (a \Leftrightarrow b) \wedge (b \Leftrightarrow c) \wedge (c \Leftrightarrow a)$. □

Exercice 8 (Équivalence) *Donner les tables de vérités des formules suivantes :*

1. $p \wedge (p \vee q)$.
2. $\neg p \wedge \neg q$.
3. $\neg(p \wedge q)$.
4. $\neg(p \vee q)$.
5. $p \vee (p \wedge q)$.
6. $\neg p \vee \neg q$.
7. p.

Parmi ces formules, indiquer les formules équivalentes . □

Exercice 9 (Équivalence) *Donner les tables de vérités des formules suivantes :*

1. $(p \Rightarrow q) \wedge (q \Rightarrow p)$.
2. $p \Rightarrow q$.
3. $p \Leftrightarrow q$.
4. $\neg q \Rightarrow \neg p$.
5. $(p \wedge q) \vee (\neg p \wedge \neg q)$.

Parmi ces formules, Indiquer les formules équivalentes. □

Exercice 10 (Équivalence) *Parmi les formules suivantes, indiquer celles qui sont équivalentes à* $p \Rightarrow q \vee r$ *?*

1. $q \wedge \neg r \Rightarrow p$.
2. $p \wedge \neg r \Rightarrow q$.
3. $\neg q \wedge \neg r \Rightarrow \neg p$.
4. $q \vee \neg p \vee r$.

□

Exercice 11 (Modèle d'un ensemble de formules,*) *Montrer qu'une assignation est un modèle d'un ensemble de formules, si et seulement si elle est un modèle de la conjonction des formules de l'ensemble.* □

Exercice 12 (Lois de simplification) *Prouver que pour tout x, y :*
- $x \vee (x \wedge y) \equiv x$.
- $x \wedge (x \vee y) \equiv x$.
- $x \vee (\neg x \wedge y) \equiv x \vee y$.
- $x \wedge (\neg x \vee y) \equiv x \wedge y$.

□

Exercice 13 (Simplification de formule,*) *Montrer par simplification que la formule suivante est une tautologie.*

$$(a \vee b) \wedge (\neg b \vee c) \Rightarrow (a \vee c)$$

□

Exercice 14 (Modèles et formes normales) *Soit A la formule suivante :*

$$(((a \Rightarrow \neg b) \Leftrightarrow \neg c) \wedge (c \vee d)) \wedge (a \Leftrightarrow d).$$

1. *A est-elle une tautologie ? (justifier)*
2. *A est-elle une contradiction ? (justifier)*
3. *Donner la forme normale conjonctive de A (détailler).*
4. *Donner la forme normale disjonctive de A (détailler).*

□

Exercice 15 (Récurrence,*) *Montrer que toute formule construite avec une seule variable, uniquement les connecteurs binaires $\vee$ et $\wedge$ et sans négation est équivalente à une formule de taille* 0. □

Exercice 16 (Algèbre de Boole) *À l'aide de tables de vérité, indiquer pour les opérations suivantes $\Rightarrow, \Leftrightarrow$ si elles sont commutatives, associatives, idempotentes, transitives.* □

Exercice 17 (Algèbre de Boole) *Montrer qu'une formule avec une seule variable, notée p, est équivalente à l'une des formules,* $0, 1, p, \neg p$. □

Exercice 18 (Algèbre de Boole) *Donner* 16 *formules telles que toute formule avec deux variables, notées p et q, soit équivalente à l'une des* 16 *formules.*

□

Exercice 19 (Fonction booléenne) *Indiquer en fonction de k combien il y a de fonctions booléennes à k arguments.* □

Exercice 20 (Conséquence) *Au cours de l'enquête dont il est chargé, l'adjudant Tifrice effectue le raisonnement suivant :*
- *Si le meurtre a eu lieu le jour, le meurtrier est l'ami de la victime*
- *Or le meurtre a eu lieu la nuit*

Donc le meurtrier n'est pas l'ami de la victime. Le raisonnement de l'adjudant Tifrice est-il correct ? Pour cela nous procédons en trois étapes :

1. *Formalisation des faits.*
2. *Formalisation du raisonnement permettant de déduire la conclusion à partir des hypothèses.*
3. *Démontrer que le raisonnement est correct ou non.*

□

Exercice 21 (Conséquence) *Pinocchio, Quasimodo et Roméo s'apprêtent à chanter en canon. Ils décident entre eux que :*

1. *Si Pinocchio ne chante pas alors Quasimodo chantera.*
2. *Si Quasimodo chante alors Pinocchio et Roméo chanteront.*
3. *Si Roméo chante alors Quasimodo ou Pinocchio, l'un des deux au moins, ne chantera pas.*

Peut-on en conclure que Pinocchio chantera ? Justifier votre réponse en formalisant le raisonnement. □

Exercice 22 (Conséquence) *Formaliser les énoncés suivants en dégageant les connecteurs logiques.*

(a) *Si Pierre est rentré chez lui, alors Jean est allé au cinéma.*

(b) *Marie est à la bibliothèque ou Pierre est rentré chez lui ou les deux.*

(c) *Si Jean est allé au cinéma, alors Marie est à la bibliothèque ou Pierre est rentré chez lui ou les deux.*

(d) *Marie n'est pas à la bibliothèque et Jean est allé au cinéma.*

(e) *Pierre est rentré chez lui.*

Le dernier énoncé est-il conséquence des énoncés qui le précèdent ? □

Exercice 23 (Formes normales) *Pour chaque formule donnée ci-après, donner sa forme normale disjonctive et prouver si elle est ou non satisfaisable (en donnant si besoin un modèle de la formule).*

— $\neg(a \Leftrightarrow b) \vee (b \wedge c) \Rightarrow c$.

— $(a \Rightarrow b) \wedge (b \Rightarrow \neg a) \wedge (\neg a \Rightarrow b) \wedge (b \Rightarrow a)$.

□

Exercice 24 (Formes normales) *Soit A la formule* $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$. *Transformer A en une fnc. La formule A est-elle valide ?*

□

Exercice 25 (Formes normales,)** *Nous revenons sur les preuves des transformations en forme normale.*

1. *Montrer que l'application de l'élimination des équivalences, des implications et de déplacement des négations sur une formule donne une formule en forme normale.*
2. *Montrer aussi que quel que soit l'ordre d'application de l'élimination des équivalences, des implications et de déplacement des négations sur une formule, l'algorithme de transformation en forme normale termine.*

□

Exercice 26 (Conséquence et formes normales) *L'adjudant Tifrice enquête une nouvelle fois. Ses hypothèses sont les suivantes :*

— *Si Jean n'a pas rencontré Pierre l'autre nuit, c'est que Pierre est le meurtrier ou que Jean est un menteur*

— *Si Pierre n'est pas le meurtrier, alors Jean n'a pas rencontré Pierre l'autre nuit et le crime a eu lieu après minuit*

— *Si le crime a eu lieu après minuit, alors Pierre est le meurtrier ou Jean est un menteur.*

Il en conclut que Pierre est le meurtrier. Son raisonnement est-il correct ? Donner la réponse en construisant la conjonction des hypothèses et de la négation de la conclusion *et en mettant cette conjonction sous forme de somme de monômes. Nous rappelons que le raisonnement est incorrect si et seulement si un des monômes de la somme ne comporte pas de littéraux complémentaires : ce monôme donne un modèle des hypothèses, qui est un contre-modèle de la conclusion.*

□

Exercice 27 (Formalisation, somme de monômes,*) *Nous nous trouvons sur une île dont les indigènes sont répartis en deux tribus, les Purs et les Pires. Les Purs disent toujours la vérité tandis que les Pires mentent toujours. Nous rencontrons deux indigènes, Aha et Bébé*[4].

(a) *Aha déclare : « L'un de nous au moins est un Pire ». Pouvons-nous en déduire ce que sont Aha et Bébé ?*

(b) *Aha déclare : « L'un de nous deux au plus est un Pire ». Pouvons-nous en déduire ce que sont Aha et Bébé ?*

(c) *Aha déclare :« Nous sommes tous les deux de la même tribu ». Pouvons-nous en déduire ce que sont Aha et Bébé ?*

□

Exercice 28 (Ensemble complet et incomplet de connecteurs,)** *Un ensemble de constantes et de connecteurs est dit* complet, *si toute fonction booléenne est exprimable avec ces constantes et ces connecteurs. D'après le théorème 1.6.3 page 29, l'ensemble* $\{0, 1, -, +, \bullet\}$ *est complet.*

1. *Montrer que l'ensemble* $\{-, +\}$ *est complet.*
   Indication : il suffit de montrer que $\{0, 1, \bullet\}$ *sont définissables à l'aide de* $-$ *et* $+$.

4. Cette énigme provient du livre de Raymond M. Smullyan : « Quel est le titre de ce livre ? », qui contient bien d'autres problèmes amusants à propos de ces indigènes.

2. *Montrer que l'ensemble* $\{0, \Rightarrow\}$ *est complet.*
3. *Soit* $|$ *l'opération suivante :* $x \mid y$ *est vrai si et seulement si ni l'un, ni l'autre ne sont vrais, autrement dit* $x \mid y = 1$ *si et seulement si* $x = 0$ *et* $y = 0$*. Cette opération est aussi appelée* ni *car* $x \mid y$ *est vrai si et seulement si ni x, ni y ne sont vrais.*

   Montrer que cette opération est complète.
4. *(**) Montrer que l'ensemble* $\{0, 1, +, \bullet\}$ *est incomplet.*

   Indication : nous trouvons une propriété caractéristique des fonctions booléennes définies avec ces opérations et qui n'est pas vérifiée par toute fonction booléenne.

   Nous appelons monotone, une fonction f *telle que si* $a_1 \leq b_1, \ldots a_n \leq b_n$ *alors* $f(a_1, \ldots a_n) \leq f(b_1, \ldots b_n)$*.*

   Montrer que toute fonction booléenne définie avec $\{0, 1, +, \bullet\}$ *est monotone.*

   Proposer une fonction booléenne non monotone.
5. *(*) L'ensemble* $\{1, \Rightarrow\}$ *est-il complet ?*
6. *(**) L'ensemble* $\{0, \Leftrightarrow\}$ *est-il complet ?*

□

Exercice 29 (Dualité, récurrence,)** *Considérons des formules dont les opérations sont limitées à* $0, 1, \neg, \wedge, \vee$*. Nous rappelons que la formule duale d'une formule est obtenue en échangeant les conjonctions et les disjonctions ainsi que les* 0 *et les* 1 *et que nous notons* A^* *la duale de la formule A.*

1. *Montrer que si 2 formules sont équivalentes, il en est de même de leurs duales. Nous donnons une indication qui réduit la question à une récurrence simple. Soit v une assignation. Soit* $\overline{v}$ *l'assignation complémentaire de v : pour toute variable x,* $\overline{v}(x) = \overline{v(x)}$*. Par exemple l'assignation complémentaire de* $a = 1, b = 0$ *est* $a = 0, b = 1$*.*

   (a) *Montrer que pour toute formule A :* $[A^*]_v = \overline{[A]_{\overline{v}}}$*.*

   (b) *En déduire que si* 2 *formules sont équivalentes, il en est de même de leurs duales.*
2. *En déduire que si une formule est valide, sa duale est contradictoire.*

□

Exercice 30 (Représentation canonique,*)** *Soit* $x_{i(0 \leq i \leq n)}$ *une liste de n variables distinctes. Dans cet exercice, les variables des formules sont toutes dans cette liste. À cette liste et à une formule A, nous associons la fonction booléenne f à n arguments définie par* $f(x_1, \ldots, x_n) = A$*. Cette notation signifie que pour* $a_1, \ldots a_n \in \mathbb{B}$*, la valeur de* $f(a_1, \ldots, a_n)$ *est la valeur de la formule A quand* $x_1 = a_1, \ldots, x_n = a_n$*.*

1. *Combien y a-t-il de classes de formules ?*
2. *Notons par* $\oplus$ *le ou-exclusif. Nous avons* $x \oplus y = 1$ *si et seulement soit* $x = 1$*, soit* $y = 1$ *mais pas* $x = y = 1$ *(d'où le nom de ou-exclusif). Nous pouvons aussi définir cette opération par* $x \oplus y = \neg(x \Leftrightarrow y)$*. Il est clair que cette opération est commutative, associative, que l'opération* $\wedge$ *est distributive sur* $\oplus$ *et que* 0 *est l'élément neutre de* $\oplus$*. Exprimer la négation, la disjonction, l'implication et l'équivalence comme somme ou-exclusive de* 1 *et de conjonctions de variables.*
3. *Montrer que toute formule booléenne est équivalente à une somme ou-exclusive de 1 et de produits de variables.*
4. *Par convention* 1 *est considéré comme le produit de zéro variable et* 0 *comme la somme ou-exclusive de zéro produit de variables. Montrer que toute formule booléenne est équivalente à une somme ou-exclusive de produits dont les ensembles de variables sont tous distincts.*
5. *Fixons un ordre arbitraire des n variables utilisées dans les formules (lorsque les variables sont des identificateurs, cet ordre est souvent l'ordre lexicographique). Une somme ou-exclusive de produits de variables est canonique (forme de Reed-Muller) si et seulement si :*
   — *ses produits ne comportent pas de variables répétées et dans un produit les variables sont ordonnées de gauche à droite dans l'ordre croissant,*
   — *les produits sont ordonnés de gauche à droite dans l'ordre lexicographique décroissant déduit de l'ordre des variables.*

   Montrer que toute formule booléenne est équivalente à une somme ou-exclusive canonique. En déduire l'unicité de cette forme canonique.

□

Chapitre 2

Résolution propositionnelle

Sommaire

2.1 Résolution **38**
- 2.1.1 Définitions 38
- 2.1.2 Cohérence 40
- 2.1.3 Complétude pour la réfutation 41
- 2.1.4 Réduction d'un ensemble de clauses 42

2.2 Stratégie complète **43**
- 2.2.1 Algorithme de la stratégie complète 43
- 2.2.2 Arrêt de l'algorithme 44
- 2.2.3 Le résultat de l'algorithme est équivalent à l'ensemble initial de clauses 45

2.3 Algorithme de Davis-Putnam-Logemann-Loveland **45**
- 2.3.1 Suppression des clauses contenant des littéraux isolés 46
- 2.3.2 Résolution unitaire 46
- 2.3.3 Suppression des clauses valides 47
- 2.3.4 L'algorithme 47
- 2.3.5 Solveurs SAT 49

2.4 Exercices **52**

LES tables de vérité et les transformations en sommes de monômes ou produits de clauses permettent de répondre aux questions : une formule est-elle valide ? et un raisonnement est-il correct ? Cependant le temps de réponse de ces méthodes augmente exponentiellement avec le nombre de variables. Pour illustrer cela, considérons la conséquence suivante :

$$a \Rightarrow b, b \Rightarrow c, c \Rightarrow d, d \Rightarrow e, e \Rightarrow f, f \Rightarrow g, g \Rightarrow h, h \Rightarrow i, i \Rightarrow j \models a \Rightarrow j$$

En utilisant une table de vérité il faut tester $2^{10} = 1024$ lignes. Or par *déduction*, en utilisant la transitivité de l'implication, nous savons immédiatement que le raisonnement ci-dessus est correct. Ainsi, en logique propositionnelle, les déductions peuvent s'avérer très utiles, voire nécessaires lorsque la taille du problème à traiter est importante. Nous étudions maintenant un système de déduction très simple. Dans ce système, les hypothèses et les formules déduites ne sont pas des formules quelconques mais des *clauses*, c'est-à-dire des disjonctions de littéraux. De plus, le système comporte une seule règle appelée *la résolution*.

Plan : Nous introduisons d'abord la méthode de *résolution* inventée en 1965 par J. Alan Robinson [23] dans le cadre plus général de la recherche de preuve au premier ordre. Nous présentons ensuite une procédure pour calculer tous les résolvants possibles, c'est-à-dire toutes les applications possibles de la règle de résolution : la *stratégie complète*. Puis, nous étudions l'algorithme `DPLL`, proposé dans les années soixante par Martin Davis, Hilary Putnam, George W. Logemann et Donald W. Loveland [10, 9]. En utilisant la notion de résolution, des méthodes de simplifications et le branchement/retour arrière, cet algorithme prouve efficacement si une formule a un modèle ou non. Bien qu'inventé il y a plus de quarante ans, il est toujours à la base de la plupart des solveurs SAT complets actuels. Notons que dans ce chapitre nous adoptons la notation booléenne plus concise.

2.1 Résolution

Nous souhaitons prouver que $a+c$ est une conséquence de $a+b$ et $\overline{b}+c$, c'est-à-dire $a+b,\ \overline{b}+c \models a+c$. En utilisant la propriété 1.2.27 page 17, il suffit de démontrer que la formule $(a+b).(\overline{b}+c) \Rightarrow (a+c)$ est valide. Pour cela, nous pouvons utiliser une table de vérité (8 lignes) ou alors une simplification de formules comme dans l'exercice 13 page 33. Ces deux méthodes sont fastidieuses même pour des formules aussi simples (3 variables uniquement). En utilisant la règle de résolution présentée dans ce chapitre, nous déduisons directement la clause $a+c$ à partir des deux clauses $a+b$ et $\overline{b}+c$.

Nous donnons maintenant la définition de la résolution et certaines propriétés utiles qui en dérivent. Nous prouvons ensuite la *cohérence* et la *complétude pour la réfutation* de notre système basé sur la déduction. La cohérence nous permet d'affirmer que si une clause C est obtenue par résolution à partir d'un ensemble Γ de clauses alors elle en est conséquence, c'est-à-dire tout modèle de Γ est modèle de C. La complétude pour la réfutation nous permet d'affirmer que si $\perp$ est conséquence de Γ, autrement dit si Γ est insatisfaisable, alors $\perp$ peut être retrouvé par résolution à partir de Γ.

2.1.1 Définitions

Dans un premier temps nous introduisons les notions nécessaires à la définition de la résolution. Nous rappelons que les notions de clauses et littéraux sont définies dans la section 1.4 du chapitre précédent (page 22). Comme nous le verrons dans la suite deux clauses qui ont le même ensemble de littéraux jouent le même rôle dans la résolution. Aussi, nous définissons deux clauses égales comme étant deux clauses avec le même ensemble de littéraux.

Définition 2.1.1 (Éléments d'une clause, clauses égales et sous-clauses)
- *Un littéral est* élément d'une clause*, s'il est élément de l'ensemble des littéraux de la clause.*
- *Une clause A est* incluse dans une clause *B, si tous les littéraux de la clause A sont éléments de la clause B. Dans ce cas, A est une* sous-clause *de B.*
- *Deux clauses sont* égales *si elles ont le même ensemble de littéraux.*

Exemple 2.1.2 *Nous illustrons les notions introduites par des exemples simples :*
- *Le littéral p est élément de la clause $\overline{q}+p+r+p$.*
- *La clause $p+\overline{q}$ est incluse dans la clause $\overline{q}+p+r+p$.*
- *Enlever le littéral p de la clause $\overline{q}+p+r+p$ donne la clause $\overline{q}+r$, enlever le littéral p de la clause $p+p+p$ donne la clause $\perp$.*
- *Ajouter le littéral r à la clause p donne la clause $p+r$, ajouter le littéral p à la clause $\perp$ donne la clause p.*
- *Les clauses $p+\overline{q}$, $\overline{q}+p$, et $p+\overline{q}+p$ sont égales.*

Notation : Nous notons $s(A)$ l'ensemble des littéraux de la clause A. Par convention $\perp$ est la clause vide et $s(\perp)=\emptyset$.

Exemple 2.1.3 $s(\overline{q}+p+r+p+\overline{p}) = \{\overline{q},p,r,\overline{p}\}$.

Dans l'exercice 37 page 53 nous demandons de formaliser ce qu'est une clause et son ensemble de littéraux. Nous introduisons la notion de littéral *complémentaire*, noté L^c pour un littéral L, afin de manipuler dans nos raisonnements uniquement des littéraux. En effet, la négation ne suffit pas car $\overline{\overline{L}}$ est équivalent à L mais n'est pas un littéral : un littéral est une variable ou la négation d'une variable mais la double négation d'une variable n'est pas un littéral.

Définition 2.1.4 (Littéral complémentaire) *Notons L^c le* littéral complémentaire *d'un littéral L, défini par :*
- *Si L est une variable, L^c est la négation de L.*
- *Si L est la négation d'une variable, L^c est obtenue en enlevant la négation de L.*

Exemple 2.1.5 $x^c = \overline{x}$ *et* $\overline{x}^c = x$.

Nous présentons le calcul de résolvant entre deux clauses, ce qui constitue l'unique règle du système de résolution.

Définition 2.1.6 (Résolvant) *Soient A et B deux clauses. La clause C est un résolvant de A et B si et seulement s'il y a un littéral L tel que $L \in s(A)$, $L^c \in s(B)$ et $s(C) = (s(A)-\{L\}) \cup (s(B)-\{L^c\})$. Nous utilisons la représentation suivante lorsque C est un résolvant de A et B :*

$$\frac{A \qquad B}{C}$$

Si la clause C est un résolvant de A et B, alors nous disons que C est engendrée *par A et B, et que A et B sont les* parents *de la clause C.*

Exemple 2.1.7 *Nous proposons quelques exemples de résolution :*

Propriété 2.1.8 *Si l'un des parents d'un résolvant est valide, le résolvant est valide ou contient l'autre parent.*

Preuve : Cette preuve est demandée dans l'exercice 39 page 53. □

Étant données deux clauses A et B, la formule $A+B$ n'est pas une clause si l'un des deux opérandes de la disjonction est la clause vide. Par exemple $\perp + p$ n'est pas une clause, bien que les deux opérandes de la disjonction soient des clauses. Aussi nous introduisons un nouvel opérande, $\tilde{+}$, afin de combiner la disjonction et l'élimination de $\perp$. Cet opérande va nous permettre de simplifier la définition de résolvant.

Définition 2.1.9 ($C\tilde{+}D$) *Soient C et D deux clauses. Nous notons $C\tilde{+}D$ la clause suivante :*
- *Si $C = \perp$ alors $C\tilde{+}D = D$,*
- *sinon si $D = \perp$ alors $C\tilde{+}D = C$ sinon $C\tilde{+}D = C + D$.*

Ajouter le littéral L à la clause C, c'est construire $C\tilde{+}L$. Munis de cette définition, nous pouvons reformuler plus simplement la règle de résolution.

Définition 2.1.10 (Résolvant) *Soient A et B deux clauses. La clause C est un* résolvant *de A et B si et seulement s'il y a un littéral L tel que :*
- *L est élément de la clause A, L^c est élément de la clause B*
- *C est égale à la clause $A'\tilde{+}B'$ où $A' = A - \{L\}$ est obtenue en enlevant L de A et $B' = B - \{L^c\}$ est obtenue en enlevant L^c de B.*

La combinaison de résolutions successives permet de générer de nouvelles clauses. Une clause issue de ces résolvants sera une preuve par résolution de cette clause à partir d'un ensemble de clauses.

Définition 2.1.11 (Preuve) *Soient Γ un ensemble de clauses et C une clause. Une* preuve *de C à partir de Γ est une liste de clauses qui se termine par C. Toute clause de la preuve est égale à un élément de Γ ou est un résolvant de deux clauses la précédant dans la preuve. La clause C est déduite de Γ, notée $\Gamma \vdash C$, s'il y a une preuve de C à partir de Γ.*

Exemple 2.1.12 *Soit Γ l'ensemble de clauses $\overline{p}+q, p+\overline{q}, \overline{p}+\overline{q}, p+q$. Nous montrons que $\Gamma \vdash \perp$ par la preuve suivante :*

Nous pouvons aussi voir une preuve comme un *arbre* dont les feuilles sont les hypothèses et dont les sommets internes sont obtenus par construction des résolvants.

Définition 2.1.13 (Taille de preuve) *Une preuve P de C à partir d'un ensemble de clauses Γ est de taille n si elle contient n lignes.*

La taille de la preuve donnée dans l'example 2.1.12 page précédente est 8, ce qui n'est pas forcément intuitif sur la représentation en arbre. Nous utiliserons les deux propriétés énoncées ci-dessous souvent de manière implicite dans les preuves.

Propriété 2.1.14 (Monotonie et composition) *Soient Γ, Δ deux ensembles de clauses et A, B deux clauses.*

1. *Monotonie de la déduction : Si $\Gamma \vdash A$ et si Γ est inclus dans Δ alors $\Delta \vdash A$*
2. *Composition des déductions : Si $\Gamma \vdash A$, $\Gamma \vdash B$ et si C est un résolvant de A et B alors $\Gamma \vdash C$.*

Preuve : Demandée dans l'exercice 38 page 53. □

2.1.2 Cohérence

La cohérence de la résolution signifie que si une clause C est obtenue après une à plusieurs applications de la règle de résolution à un ensemble de clauses Γ alors tout modèle de Γ est aussi modèle de C (autrement dit, C est une conséquence de Γ). La première partie de la preuve consiste à démontrer la cohérence d'une application de la règle de résolution (théorème 2.1.15). Nous généralisons ensuite par induction dans le théorème 2.1.16 page suivante.

Théorème 2.1.15 (Cohérence de la règle de résolution) *Si C est un résolvant de A et B alors $A, B \models C$.*

Preuve : Supposons que C soit un résolvant de A et B. Par définition il y a un littéral L tel que $L \in s(A), L^c \in s(B), s(C) = (s(A) - \{L\}) \cup (s(B) - \{L^c\})$. Soit v une assignation modèle de A et B, c'est-à-dire, $[A]_v = 1$ et $[B]_v = 1$. Montrons que v est modèle de C. Nous distinguons deux cas possibles :

□

Théorème 2.1.16 (Cohérence de la déduction) *Soient Γ un ensemble de clauses et C une clause. Si $\Gamma \vdash C$ alors $\Gamma \models C$.*

Preuve : Supposons que $\Gamma \vdash C$. Par définition, il existe une preuve P de C à partir de Γ. Supposons que pour toute preuve de D à partir de Γ, *plus courte* que P, nous avons $\Gamma \models D$. Montrons que $\Gamma \models C$. Nous avons deux cas possibles :

□

Corollaire 2.1.17 *D'après le théorème 2.1.16, si $\Gamma \vdash \bot$ alors $\Gamma \models \bot$, autrement dit Γ est insatisfaisable.*

2.1.3 Complétude pour la réfutation

La complétude pour la réfutation est la propriété suivante : si Γ est un ensemble insatisfaisable de clauses, alors il y a une preuve de la clause vide à partir de Γ. Nous montrons dans la suite de cette section la complétude pour la réfutation lorsque Γ est un ensemble *fini* de clauses.

Nous prouvons la complétude réfutationnelle par récurrence sur le nombre de variables. Pour cela, nous construisons à partir de Γ deux ensembles de clauses contenant tous deux une variable en moins. Plus précisément, nous construisons les ensembles $\Gamma[L := 1]$ et $\Gamma[L := 0]$ en assignant un littéral L de Γ à vrai et à faux, respectivement. Nous étudions ensuite les propriétés de ces deux ensembles par rapport à Γ (lemmes 2.1.22 page suivante et 2.1.23 page suivante). Ces propriétés seront utilisées lors du pas de récurrence.

Définition 2.1.18 ($\Gamma[L := 1]$) *Soient Γ un ensemble de clauses et L un littéral. $\Gamma[L := 1]$ est l'ensemble de clauses obtenu en supprimant les clauses dont L est élément et en enlevant L^c des autres clauses. Nous posons $\Gamma[L := 0] = \Gamma[L^c := 1]$.*

$\Gamma[L := x]$ est égal à l'ensemble de clauses obtenu en donnant à L la valeur x ($x = 0$ ou $x = 1$) et en simplifiant le résultat obtenu.

Exemple 2.1.19 *Soit Γ l'ensemble de clauses $\overline{p}+q, \overline{q}+r, p+q, p+r$. Nous avons :*

— $\Gamma[p := 1] =$

— $\Gamma[p := 0] =$

Afin de donner une intuition de comment calculer $\Gamma[p := 1]$ et $\Gamma[p := 0]$, nous considérons le produit de clauses suivant $(\overline{p}+q)(\overline{q}+r)(p+q)(p+r)$ et observons :

— $(\overline{1}+q)(\overline{q}+r)(1+q)(1+r) =$

— $(\overline{0}+q)(\overline{q}+r)(0+q)(0+r) =$

Nous notons $v[L := 1]$ l'assignation qui donne à L la valeur 1, à L^c la valeur 0 et ne change pas la valeur des autres littéraux.

Définition 2.1.20 ($v[L := 1]$) *Soit une assignation v, l'assignation $v[L := 1]$ est l'assignation identique à v sauf éventuellement pour x la variable de L. Si $L = x$ alors $v[L := 1](x) = 1$, si $L = \overline{x}$ alors $v[L := 1](x) = 0$.*

Nous posons $v[L := 0] = v[L^c := 1]$. La propriété suivante se déduit du fait que si Γ a un modèle v, alors v donne soit la valeur 1 soit la valeur 0 au littéral L.

Propriété 2.1.21 *Soient Γ un ensemble de clauses et L un littéral. Γ a un modèle si et seulement si $\Gamma[L := 1]$ ou $\Gamma[L := 0]$ a un modèle.*

Lemme 2.1.22 *Soient Γ un ensemble de clauses, C une clause et L un littéral. Si $\Gamma[L := 1] \vdash C$ alors $\Gamma \vdash C$ ou $\Gamma \vdash C \tilde{+} L^c$.*

Rappel : L'opération $\tilde{+}$ est introduite dans la définition 2.1.9 page 39.

Lemme 2.1.23 *Soient Γ un ensemble de clauses, C une clause et L un littéral. Si $\Gamma[L := 0] \vdash C$ alors $\Gamma \vdash C$ ou $\Gamma \vdash C \tilde{+} L$.*

Preuve : Supposons $\Gamma[L := 0] \vdash C$. Puisque $\Gamma[L := 0] = \Gamma[L^c := 1]$ et que $L^{cc} = L$, d'après le lemme 2.1.22 nous avons $\Gamma \vdash C$ ou $\Gamma \vdash C \tilde{+} L$. □

Théorème 2.1.24 *Soit Γ un ensemble fini de clauses. Si Γ est insatisfaisable alors $\Gamma \vdash \bot$.*

Preuve : Supposons que Γ est insatisfaisable. Soit n le nombre de variables de Γ. Nous montrons que $\Gamma \vdash \bot$ par récurrence sur n. Supposons que pour tout ensemble Δ de clauses insatisfaisable avec moins de n variables, nous avons $\Delta \vdash \bot$.

Cas de base : Supposons que n soit nul. Donc $\Gamma = \emptyset$ ou $\Gamma = \{\bot\}$. Le premier cas est impossible, car l'ensemble vide est valide (toute assignation en est modèle). Donc $\Gamma = \{\bot\}$ et par suite $\Gamma \vdash \bot$.

Induction : Supposons que n soit non nul. Soit x une variable figurant dans Γ. D'après la propriété 2.1.21, $\Gamma[x := 0]$ et $\Gamma[x := 1]$ sont insatisfaisables. Puisque la variable x ne figure pas dans ces deux ensembles de clauses, l'hypothèse de récurrence s'applique, donc : $\Gamma[x := 0] \vdash \bot$ et $\Gamma[x := 1] \vdash \bot$. Des lemmes 2.1.22 et 2.1.23, nous déduisons soit $\Gamma \vdash \bot$, soit $\Gamma \vdash \overline{x}$ et $\Gamma \vdash x$. Dans le premier cas, la preuve est terminée. Dans le deuxième cas, puisque $\bot$ est un résolvant de $\overline{x}$ et x, nous avons également $\Gamma \vdash \bot$.

□

Corollaire 2.1.25 *Soit Γ un ensemble fini de clauses. Γ est insatisfaisable si et seulement si $\Gamma \vdash \bot$.*

Preuve : Ce corollaire est une conséquence immédiate du lemme ci-dessus et de la remarque faite au début du paragraphe 2.1.3 page précédente. □

2.1.4 Réduction d'un ensemble de clauses

Réduire un ensemble de clauses, c'est lui enlever les clauses valides et les clauses contenant une *autre* clause de l'ensemble. Un ensemble de clauses est réduit s'il n'est plus réductible.

Définition 2.1.26 (Ensemble de clauses réduit) *Un ensemble de clauses est* réduit *s'il ne contient aucune clause valide et aucune clause n'est incluse dans une autre clause.*

Exemple 2.1.27 *La réduction de l'ensemble de clauses $\{p+q+\overline{p}, p+r, p+r+\overline{s}, r+q\}$ donne l'ensemble réduit :*

Propriété 2.1.28 *Un ensemble de clauses E est équivalent à l'ensemble de clauses réduit obtenu à partir de E.*

Preuve :

□

2.2 Stratégie complète

Pour savoir si un ensemble fini Γ de clauses est contradictoire, il y a une méthode simple mais peu efficace : construire *toutes* les clauses que nous pouvons déduire de Γ. D'après le corollaire 2.1.25 page ci-contre, Γ est contradictoire si et seulement si la clause vide a été construite. Il faut préciser ce que nous entendons par construire toutes les clauses déduites de Γ. Si nous considérons comme différentes les clauses qui s'écrivent différemment, deux clauses ont une infinité de résolvants. Par exemple les clauses $p+\overline{q}$ et $r+q$ ont comme résolvants la clause $p+r$ mais aussi les clauses $r+p$, $p+r+p$, $p+\ldots+p+r+\cdots+r$. Donc l'ensemble des clauses déduites de Γ est infini. Aussi, il faut, comme précédemment, considérer comme égales deux clauses, qui ont le même ensemble de littéraux (voir définition 2.1.1 page 38) : avec cette *définition de l'égalité de deux clauses*, l'ensemble des résolvants de deux clauses est fini ainsi que l'ensemble des clauses déduites de Γ. Supposons que l'ensemble des littéraux de Γ ait n éléments, comme les clauses déduites de Γ ne comportent que des littéraux de Γ (voir exercice 40 page 53), il y a donc au plus 2^n clauses déduites de Γ. Pour simplifier la suite, comme ci-dessus, nous confondons une clause et son ensemble de littéraux.

2.2.1 Algorithme de la stratégie complète

Soit Γ un ensemble de clauses. Pour obtenir l'ensemble des clauses minimales déduites de Γ, on construit deux suites d'ensembles de clauses $\Delta_{i(i\geq 0)}$ et $\Theta_{i(i\geq 0)}$ en s'arrêtant au premier indice k tel que $\Delta_k = \emptyset$.

1. Δ_0 est obtenu en réduisant Γ et Θ_0 est l'ensemble vide.
2. Δ_{i+1} est obtenu de la façon suivante
   (a) Nous construisons l'ensemble des résolvants entre les clauses de Δ_i et celles de $\Delta_i \cup \Theta_i$ puis nous réduisons cet ensemble.
   (b) Nous enlevons de cet ensemble de résolvants, les clauses qui contiennent une clause de $\Delta_i \cup \Theta_i$.
3. Θ_{i+1} est obtenu en enlevant de l'ensemble $\Delta_i \cup \Theta_i$ les clauses qui contiennent une clause de Δ_{i+1}.

Nous illustrons notre algorithme par deux exemples.

Exemple 2.2.1 *Nous appliquons l'algorithme à l'ensemble de clauses $a+b+\overline{a}, a+b, a+b+c, a+\overline{b}, \overline{a}+b, \overline{a}+\overline{b}$. L'algorithme effectue alors le calcul suivant. L'ensemble Δ_0 comprend les clauses :*

1	$a+b$
2	$a+\overline{b}$
3	$\overline{a}+b$
4	$\overline{a}+\overline{b}$

L'ensemble Θ_0 est vide. L'algorithme calcule les résolvants non valides suivants :

5	a	*résolvant de 1 et 2*
6	b	*résolvant de 1 et 3*
7	$\overline{b}$	*résolvant de 2 et 4*
8	$\overline{a}$	*résolvant de 3 et 4*

Donc l'ensemble Δ_1 comprend les mêmes clauses que nous numérotons de 5 à 8.
L'ensemble Θ_1 est vide.
L'ensemble Δ_2 comprend la clause :

9	$\perp$	*de 5 et 8*

Car l'algorithme obtient deux fois $\perp$.
L'ensemble Θ_2 est vide.
L'ensemble Δ_3 est vide.
L'ensemble Θ_3 est égal à Δ_2.
Nous notons que l'algorithme a construit la preuve suivante :

1	$a+b$	
2	$a+\overline{b}$	
5	a	*résolvant de 1 et 2*
3	$\overline{a}+b$	
4	$\overline{a}+\overline{b}$	
8	$\overline{a}$	*résolvant de 3 et 4*
9	$\perp$	*résolvant de 5 et 8*

Nous pouvons utiliser la représentation ci-dessous sous forme de tableau représentant la construction de ces ensembles.

k	Δ_k	Θ_k	$\Delta_k \cup \Theta_k$	*Résolvants de* Δ_k *et* $\Delta_k \cup \Theta_k$
0				
1				
2				
3				

Comme le montre cet exemple, les clauses de Δ_i ont des preuves de hauteur i, et celles de Θ_i des preuves de hauteur inférieure à i.

Exemple 2.2.2 *L'application de la stratégie complète sur l'ensemble* $\{a, c, \overline{a}+\overline{b}, \overline{c}+e\}$ *nous donne l'ensemble* $\{a, c, e, \overline{b}\}$.

k	Δ_k	Θ_k	$\Delta_k \cup \Theta_k$	*Résolvants de* Δ_k *et* $\Delta_k \cup \Theta_k$
0				
1				
2				

Nous montrons que l'algorithme se termine et qu'il est correct, c'est-à-dire que l'ensemble Θ_k est l'ensemble des clauses minimales déduites de Γ. Γ est insatisfaisable si et seulement si la clause vide est élément de Θ_k.

2.2.2 Arrêt de l'algorithme

Soit n le nombre de littéraux de Γ. Les clauses éléments des Δ_i sont des clauses déduites de Γ, donc, comme nous l'avons rappelé ci-dessus, elles ne comportent que des littéraux de Γ. Par suite, l'ensemble $\bigcup_{i<k} \Delta_i$ comprend au plus 2^n clauses. D'après l'algorithme, les ensembles Δ_i sont non vides pour $i < k$. Nous montrons ci-dessous que ces ensembles sont disjoints. Par conséquent $k \leq 2^n + 1$, autrement dit l'algorithme s'arrête.

Propriété 2.2.3 *Soit* $i \leq k$. *Toute clause de* $\bigcup_{j \leq i} \Delta_j$ *contient une clause de* $\Delta_i \cup \Theta_i$.

Preuve : Nous effectuons une preuve par induction.

— Pour $i = 0$ la propriété est triviale car $\bigcup_{j \leq 0} \Delta_j = \Delta_0$ et $\Delta_i \cup \Theta_i = \Delta_0$ ($\Theta_0 = \emptyset$).

— Supposons la propriété vraie au rang i et montrons qu'elle reste vraie au rang $i+1$. Soit $C \in \bigcup_{j \leq i+1} \Delta_j$. Montrons que C contient une clause de $\Delta_{i+1} \cup \Theta_{i+1}$. Nous examinons tous les cas possibles pour C.

1. $C \in \Delta_{i+1}$. Donc C contient une clause de $\Delta_{i+1} \cup \Theta_{i+1}$.
2. $C \in \bigcup_{j \leq i} \Delta_j$. Par hypothèse de récurrence, C contient une clause $D \in \Delta_i \cup \Theta_i$. Nous distinguons deux situations pour D.
   (a) $D \in \Theta_{i+1}$. Donc C contient une clause de $\Delta_{i+1} \cup \Theta_{i+1}$.
   (b) $D \notin \Theta_{i+1}$. Par construction de Θ_{i+1}, puisque $D \in \Delta_i \cup \Theta_i$ et que $D \notin \Theta_{i+1}$, c'est que D contient une clause de Δ_{i+1}. Or C contient D, donc C contient aussi une clause de $\Delta_{i+1} \cup \Theta_{i+1}$.

□

Propriété 2.2.4 *Pour tout* $i \leq k$, *les ensembles* Δ_i *sont disjoints entre eux.*

Preuve : Nous effectuons une récurrence sur les ensembles Δ_j avec $0 \leq j \leq i$ et $i \leq k$.

Cas de base : Si $i = 0$, il n'y a qu'un seul ensemble, donc la propriété est vérifiée.

Récurrence : Soit $i < k$. Supposons que tous les ensembles Δ_j où $j \leq i$ sont disjoints entre eux.

Montrons que Δ_{i+1} est disjoint de ces ensembles.
Soit $C \in \Delta_{i+1}$. Supposons, au contraire, que $C \in \bigcup_{j \leq i} \Delta_j$. D'après la propriété précédente, C inclut une clause de $\Delta_i \cup \Theta_i$. Donc par construction de Δ_{i+1}, $C \notin \Delta_{i+1}$, contradiction. Par suite, $C \notin \bigcup_{j \leq i} \Delta_j$.

□

2.2.3 Le résultat de l'algorithme est équivalent à l'ensemble initial de clauses

Nous montrons ci-dessous que l'ensemble Θ_k est équivalent à Γ.

Propriété 2.2.5 *Pour tout $i < k$, les ensembles $\Delta_i \cup \Theta_i$ et $\Delta_{i+1} \cup \Theta_{i+1}$ sont équivalents.*

Preuve :

1. Toute clause de $\Delta_{i+1} \cup \Theta_{i+1}$ est conséquence de $\Delta_i \cup \Theta_i$. En effet toute clause de $\Delta_{i+1} \cup \Theta_{i+1}$ est élément de $\Delta_i \cup \Theta_i$ ou résolvant de deux éléments de cet ensemble, donc est conséquence de cet ensemble.
2. Toute clause de $\Delta_i \cup \Theta_i$ est conséquence de $\Delta_{i+1} \cup \Theta_{i+1}$. Soit $C \in \Delta_i \cup \Theta_i$. Nous distinguons deux cas possibles :
   (a) $C \in \Theta_{i+1}$, ainsi C est conséquence de $\Delta_{i+1} \cup \Theta_{i+1}$.
   (b) $C \notin \Theta_{i+1}$, ainsi C contient une clause de Δ_{i+1} donc est conséquence de $\Delta_{i+1} \cup \Theta_{i+1}$.

□

Propriété 2.2.6 *Les ensembles Γ et Θ_k sont équivalents.*

Preuve : Par définition Δ_0 est l'ensemble obtenu par réduction de Γ, d'après la propriété 2.1.28 page 42, ces deux ensembles sont équivalents. Puisque Θ_0 est vide, Γ est équivalent à $\Delta_0 \cup \Theta_0$. D'après la propriété 2.2.5 et par récurrence, $\Delta_0 \cup \Theta_0$ est équivalent à l'ensemble de clauses $\Delta_k \cup \Theta_k$. Puisque l'algorithme termine quand Δ_k est l'ensemble vide, les ensembles Γ et Θ_k sont équivalents. □

2.3 Algorithme de Davis-Putnam-Logemann-Loveland

La méthode de Davis-Putnam-Logemann-Loveland (`DPLL`) fut crée en 1960 par Martin Davis et Hillary Putnam, puis améliorée en 1962 par Martin Davis, George Logemann et Donald Loveland. L'algorithme `DPLL` permet de décider si un ensemble de clauses est satisfaisable. Bien que cet algorithme date de plus de 50 ans, de nombreuses variantes en sont encore étudiées, car il est bien meilleur que ceux précédemment étudiés (table de vérité, transformation en somme de monômes ou produit de sommes, stratégie complète). Nous remarquons d'abord qu'il y a deux types de transformations de formules :

1. Celles qui préservent le sens, c'est-à-dire transforment une formule en une autre formule équivalente.
2. Celles qui préservent seulement la satisfaisabilité, c'est-à-dire transforment une formule satisfaisable en une autre formule satisfaisable.

L'efficacité de l'algorithme `DPLL` vient de l'utilisation des transformations préservant seulement la satisfaisabilité, car ces transformations sont moins coûteuses que celles préservant le sens. Nous introduisons d'abord la suppression des clauses contenant des littéraux isolés, puis la résolution unitaire et enfin la simplification des clauses valides. Ces transformations constituent les briques de base de l'algorithme `DPLL`. L'idée principale de cet algorithme, partant d'un ensemble de clauses « irréductibles », est de choisir une variable et de considérer les deux situations possibles : le cas où elle est vraie et le cas où elle est fausse. Ainsi les deux ensembles de clauses obtenus peuvent éventuellement se simplifier, sinon le choix d'une nouvelle variable est nécessaire. Dans le pire des cas il s'agit de construire la table de vérité correspondante aux clauses, mais en pratique de nombreuses simplifications apparaissent et permettent à cet algorithme de conclure efficacement.

2.3.1 Suppression des clauses contenant des littéraux isolés

Comme nous ne cherchons pas à préserver l'équivalence de formule mais la satisfaisabilité, nous avons la liberté de choisir la valeur de certaines variables. En particulier une variable qui n'apparaît que sous la forme d'un littéral positif ou négatif sera dite *isolée*. Nous pouvons alors affecter la valeur adéquate à cette variable afin de rendre vraies toutes les occurrences du littéral associé, ceci sans changer la satisfaisabilité de l'ensemble de clauses considéré.

Définition 2.3.1 (Littéral isolé) *Soient Γ un ensemble de clauses et L un littéral élément d'une clause de Γ, L est un* littéral isolé *(relativement à Γ), si aucune clause de Γ ne comporte de littéral complémentaire de L.*

Lemme 2.3.2 *La suppression des clauses qui comportent un littéral isolé, préserve la satisfaisabilité.*

La preuve est demandée dans l'exercice 49 page 54.

Exemple 2.3.3 *Soit Γ l'ensemble de clauses $\{p+q+r, \overline{q}+\overline{r}, q+s, \overline{s}+t\}$.*

2.3.2 Résolution unitaire

Définition 2.3.4 *Une* clause unitaire *est une clause qui ne comporte qu'un littéral.*

Les clauses unitaires sont aussi des clauses particulières qui sont :
— soit isolées et donc seront traitées par la simplification précédente,
— soit de potentiels résolvants, dans ce cas il faut effectuer la résolution dite « unitaire » de ces clauses.

Lemme 2.3.5 *Soit Γ un ensemble de clauses. Soit L l'ensemble des littéraux des clauses unitaires de Γ. Soit Θ l'ensemble de clauses ainsi obtenu à partir de Γ :*
— *Si L comporte deux littéraux complémentaires, alors $\Theta = \{\bot\}$.*
— *Sinon Θ est obtenu ainsi :*
  - *Nous enlevons les clauses qui comportent un élément de L.*
  - *À l'intérieur des clauses restantes nous enlevons les littéraux complémentaires des éléments de L.*

Γ a un modèle si et seulement si Θ en a un.

La preuve est demandée dans l'exercice 50 page 54.

Exemple 2.3.6
— *Soit Γ l'ensemble de clauses : $p+q, \overline{p}, \overline{q}$.*
— *Soit Γ l'ensemble de clauses : $a+b+\overline{d}, \overline{a}+c+\overline{d}, \overline{b}, d, \overline{c}$.*

— *Soit* Γ *l'ensemble de clauses :* $p, q, p+r, \overline{p}+r, q+\overline{r}, \overline{q}+s$.

2.3.3 Suppression des clauses valides

Enfin la dernière transformation utilisée dans l'algorithme `DPLL` est de simplement enlever les clauses valides d'un ensemble de clauses. Ceci peut paraître une évidence mais cette transformation est une des idées centrales de cet algorithme, une fois une variable affectée à une valeur, elle est implicitement utilisée dans la suppression de clauses contenant des littéraux isolés.

Lemme 2.3.7 *Soit* Γ *un ensemble de clauses. Soit* Θ *l'ensemble de clauses obtenu en supprimant les clauses valides de* Γ. Γ *a un modèle si et seulement si* Θ *en a un.*

Preuve :

— Supposons que Γ a un modèle v, comme Θ est un sous-ensemble des clauses de Γ, v est aussi modèle de Θ. Donc, Θ a un modèle.

— Supposons que Θ a un modèle v. Soit v' une assignation de Γ telle que $v'(x) = v(x)$ pour toute variable x présente à la fois dans Γ et Θ. Soit C une clause de Γ. Si C est aussi une clause de Θ, alors v' est un modèle de C car v et v' donnent la même valeur à C. Si C n'est pas une clause de Θ, alors C est valide, en conséquence toute assignation, v' en particulier, est modèle de C. D'où, Γ a un modèle : v'.

□

2.3.4 L'algorithme

Nous donnons une version de l'algorithme `DPLL` dans la figure 2.1 page suivante dans la fonction Algo_DPLL. Nous pouvons constater dans l'algorithme que nous supprimons une seule fois les clauses valides avant de réellement commencer les simplifications. Cela se justifie par le fait que l'algorithme ne pourra pas produire de nouvelles clauses valides. En effet, il ne fait qu'enlever des littéraux des clauses initialement données. Par suite, il est inutile d'enlever des clauses valides lors de l'étape de réduction, puisqu'il n'y aura plus de telles clauses. La fonction `DPLL` teste d'abord si la clause vide a été générée, puis effectue les simplifications possibles avant de choisir une variable pour appliquer récursivement la même fonction dans le cas où cette variable est affectée à vraie et dans le cas où elle est affectée à faux.

Pour obtenir une trace de l'algorithme, nous supprimons les clauses valides, en abrégé VAL, puis nous dessinons l'arbre des appels avec l'argument de la fonction ainsi que les ensembles obtenus par les étapes 2 (réduction, en abrégé RE), 3 (enlèvement des clauses ayant des littéraux isolés, en abrégé ELI) et 4 (résolution unitaire, abrégé en RU). À cause du « ou alors », il est inutile de poursuivre la construction de cet arbre dès que la réponse vrai (attachée à un ensemble vide) a été obtenue. Dans ce cas, un modèle peut être exhibé en remontant la récursion, en tenant compte des simplifications : le modèle doit rendre vrai chaque littéral isolé ou clause unitaire trouvé.

Exemple 2.3.8 *Nous illustrons l'application de cet algorithme sur deux ensembles de clauses.*

— *Soit* Γ *l'ensemble de clauses :* $\overline{a}+\overline{b}, a+b, \overline{a}+\overline{c}, a+c, \overline{b}+\overline{c}, b+c$. *Nous donnons la trace de l'algorithme.*

bool fonction Algo_DPLL(Γ : ensemble de clauses)

0 Supprimer les clauses valides de Γ.
Si $\Gamma = \emptyset$, retourner (`vrai`).
Sinon retourner (`DPLL`(Γ))

bool fonction `DPLL`(Γ : ensemble de clauses non valides)

La fonction retourne vrai si et seulement si Γ est satisfaisable

1 **Si** $\perp \in \Gamma$, retourner(`faux`).
Si $\Gamma = \emptyset$, retourner (`vrai`).

2 Réduire Γ : il suffit d'enlever toute clause contenant une *autre* clause.

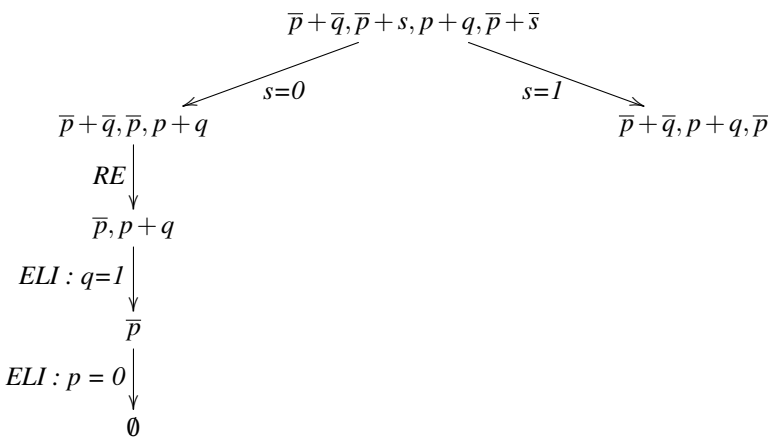
3 Enlever de Γ les clauses comportant des littéraux isolés (cf. paragraphe 2.3.1 page 46).
Si l'ensemble Γ a été modifié, aller en 1.

4 Appliquer à Γ la résolution unitaire (cf paragraphe 2.3.2 page 46).
Si l'ensemble Γ a été modifié, aller en 1.

5 Choisir x une variable quelconque de Γ
retourner (`DPLL`($\Gamma[x := 0]$) ou alors `DPLL`($\Gamma[x := 1]$))

FIGURE 2.1 – Algorithme de Davis-Putnam-Logemann-Loveland.

— *Soit Γ l'ensemble de clauses : $\overline{p}+\overline{q}, \overline{p}+s, p+q, \overline{p}+\overline{s}$. Nous donnons la trace de l'algorithme.*



Puisqu'une feuille porte l'ensemble vide, l'ensemble Γ est satisfaisable. Il est inutile de poursuivre la construction

de la branche droite.

Théorème 2.3.9 *L'algorithme Algo_DPLL est correct.*

Preuve : La réduction et les transformations 2.3.1 page 46, 2.3.2 page 46 et 2.3.3 page 47 préservent l'existence d'un modèle, donc au pas 0 et 1, l'algorithme vérifie l'invariant suivant : la valeur courante de l'ensemble de clauses de Γ a un modèle si et seulement si Γ a un modèle. Nous en déduisons immédiatement que les réponses de l'algorithme, fournies au pas 0 ou 1, sont correctes. Au pas 5, pour les mêmes raisons, la valeur courante de l'ensemble de clauses de Γ, lors de l'appel de `DPLL`, a un modèle si et seulement si Γ a un modèle. Supposons les appels récursifs corrects :

- Si `DPLL`$(\Gamma[x := 0])$ est vrai, par récurrence $\Gamma[x := 0]$ est satisfaisable donc Γ est satisfaisable (nous utilisons le résultat 2.1.21 page 42 : Γ est satisfaisable ssi $\Gamma[x := 0]$ est satisfaisable ou $\Gamma[x := 1]$ est satisfaisable), ce qui correspond à la valeur vrai de `DPLL`(Γ).
- Si `DPLL`$(\Gamma[x := 0])$ est faux, par récurrence $\Gamma[x := 0]$ est insatisfaisable. Dans ce cas, `DPLL`(Γ) vaut `DPLL`$(\Gamma[x := 1])$:
  - Supposons que `DPLL`$(\Gamma[x := 1])$ soit vrai alors par récurrence $\Gamma[x := 1]$ est satisfaisable donc Γ est satisfaisable, ce qui correspond à la valeur vrai de `DPLL`(Γ).
  - Supposons que `DPLL`$(\Gamma[x := 1])$ soit faux, alors par récurrence $\Gamma[x := 1]$ est insatisfaisable. Donc Γ est insatisfaisable, ce qui correspond à la valeur faux de `DPLL`(Γ).

□

Théorème 2.3.10 *L'algorithme Algo_DPLL se termine.*

Preuve : Au pas 0, la fonction Algo_DPLL consiste en un test, puis soit le renvoi de la valeur vrai soit l'appel de la fonction `DPLL`(Γ). Le pas 0 consiste en la suppression de clauses de l'ensemble de départ, cet ensemble étant fini, le pas 0 termine. Donc, la fonction Algo_DPLL termine si la fonction `DPLL`(Γ) termine.

Considérons un appel de la fonction `DPLL`(Γ). Le pas 1 est constitué d'un test et d'instructions élémentaires, donc il termine. Le pas 2 consiste à réduire Γ qui est un ensemble fini. La réduction consistant à supprimer des clauses, elle termine. Le pas 3 consiste à supprimer des clauses, puis éventuellement retourner au pas 1. Le nombre de clauses étant fini, soit l'algorithme termine en 1, soit le pas 4 débute. De la même manière, le pas 4 consiste à supprimer des clauses et des variables, qui sont en nombres finis, puis éventuellement retourner au pas 1. Le nombre de clauses et de variables étant fini, soit l'algorithme termine en 1, soit le pas 5 débute. Le pas 5 consiste en un ou deux appels récursifs sur un ensemble de clauses où une variable a disparu (la valeur d'une variable est fixée). Ainsi, la récurrence en 5 se termine car à chaque appel récursif le nombre de variables diminue strictement. D'où, l'algorithme Algo_DPLL termine. □

Remarque 2.3.11 (Oubli de simplifications) *L'algorithme Algo_DPLL a été donné avec les étapes de « simplification » : suppression des clauses valides (0), réduction (2), enlèvement des littéraux isolés (3) et réduction unitaire (4). L'algorithme reste correct si nous omettons ces étapes, ou si nous ne faisons qu'une partie des simplifications, ce qui est souvent le cas lorsque nous faisons une trace de l'exécution sans machine. Car sans machine il est fréquent d'oublier des simplifications, ce qui ne nuit pas car l'algorithme reste correct.*

Remarque 2.3.12 (Choix de la variable) *Un bon choix pour la variable x de l'étape (5), consiste à choisir la variable qui apparaît le plus souvent. Un meilleur choix consiste à choisir la variable qui va entraîner par la suite le plus de simplifications : il faut faire des compromis entre le temps passé à choisir la « bonne » variable et l'importance des simplifications induites par ce choix. Plusieurs heuristiques classiques de choix de variable sont présentées dans le paragraphe 2.3.5.1 page suivante.*

2.3.5 Solveurs SAT

Le problème *SAT* est un problème de décision qui consiste à déterminer si une formule propositionnelle en forme normale conjonctive admet ou non un modèle. Ce problème est le problème NP-complet de référence [5]. Généralement, seule une version de ce problème, appelée 3-SAT, où toutes les clauses de la formule sont constituées d'exactement trois littéraux, est considérée, car des *réductions linéaires* existent pour passer de SAT à 3-SAT. Un exemple de réduction SAT vers 3-SAT est étudié dans l'exercice 51 page 54.

Des programmes performants sont dédiés à la résolution du problème SAT[1]. Ces programmes sont appelés des *solveurs SAT* et sont généralement classés en deux catégories :

1. Dans ces programmes, le formalisme 3-SAT est généralement préféré au formalisme SAT, car son aspect régulier permet d'obtenir des solutions plus efficaces.

— Les solveurs SAT « complets » sont généralement des versions améliorées de l'algorithme DPLL. Parmi les solveurs SAT complets les plus utilisés à l'heure actuelle, nous pouvons citer par exemple *zchaff*, *satz*, *march* ou encore *GRASP*.
— Les solveurs SAT « incomplets » sont des algorithmes de semi-décision qui finissent par trouver un modèle à la formule, s'il existe et qui ne terminent pas dans le cas contraire. Ces algorithmes sont fondés sur des marches aléatoires dans l'espace d'état. Dans cette catégorie, nous pouvons citer les algorithmes par exemple *WalkSAT* et *GSAT*.

Nous décrivons maintenant, de manière non-exhaustive, les principales améliorations de l'algorithme DPLL utilisées dans les solveurs SAT complets.

2.3.5.1 Heuristique de branchement

Le choix de la prochaine variable à affecter a un impact important sur la taille de l'arbre de recherche développé par DPLL. Ainsi, le choix de l'heuristique de branchement est déterminant dans les algorithmes fondés sur DPLL. Les heuristiques de branchement sont souvent basées sur des arguments syntaxiques tels que la taille des clauses, le nombre d'occurences, *etc*. Parmi les heuristiques les plus connues, nous pouvons citer :

MOMS : L'heuristique *MOMS* (pour Maximum Occurences in Clauses of Minimum Size) consiste, comme son nom l'indique, à sélectionner la variable ayant le plus d'occurences dans les clauses les plus courtes. Les choix effectués par cette heuristique favorisent la propagation unitaire.

JW : L'heuristique *JW* (pour Jeroslow-Wang) utilise également la longueur des clauses. En effet, un poids est affecté à chaque littéral en fonction de la taille des clauses où il apparaît. La possibilité qu'une variable soit sélectionnée par JW est inversement proportionnelle à la somme des tailles des clauses où elle apparaît.

UP : Contrairement aux deux heuristiques précédentes, l'heuristique *UP* (pour Unit Propagation) n'est pas fondée sur des arguments syntaxiques. Cette heuristique essaie de sélectionner une variable en prévoyant à l'avance son effet sur la recherche. Elle consiste à calculer un poids pour chaque variable en fonction des simplifications qu'elle génère par propagation unitaire.

2.3.5.2 Ajout de clauses

L'idée est d'ajouter des clauses, en pré-traitement, à l'ensemble initial de clauses . Le nouvel ensemble de clauses reste équivalent à l'ensemble initial. Par exemple, nous pouvons citer la *résolution restreinte* qui consiste à ajouter des résolvants de taille bornée. Ce type de méthode permet de réduire la taille de l'arbre recherche en amenant aux contradictions plus tôt dans la recherche.

2.3.5.3 Analyse des conflits et retour-arrière non chronologique

Lorsque l'algorithme DPLL atteint une contradiction (un conflit), il revient (chronologiquement) au dernier branchement de variable effectué. Or, ce branchement n'est pas nécessairement l'une des décisions à l'origine de la contradiction. L'analyse des conflits remédie à ce problème en tenant compte des décisions à l'origine du conflit. Cette analyse permet ainsi d'effectuer un retour-arrière non-chronologique, c'est-à-dire un retour direct au niveau, plus haut dans l'arbre, qui est à l'origine du conflit sans tester les niveaux intermédiaires.

2.3.5.4 Apprentissage

L'apprentissage consiste à analyser et apprendre les raisons qui ont amené à une contradiction : lorsqu'une assignation partielle amène à une contraction, il est possible d'isoler un sous-ensemble d'assignations et un sous-ensemble de clauses, qui sont responsables du conflit. À partir de ces assignations, il est possible de construire (d'apprendre) une clause qui est impliquée par ces clauses. Cette nouvelle clause est ajoutée à l'ensemble de clauses du problème. Les assignations pertinentes sont déterminées par un graphe de dépendance entre les clauses et les assignations, appelé *graphe d'implication*. Les clauses apprises permettent de ne pas refaire plusieurs fois les mêmes « erreurs » dans l'arbre de recherche.

2.3.5.5 Redémarrage

La plupart des solveurs SAT actuels utilisent une stratégie de redémarrage, qui consiste à redémarrer la recherche en prenant en compte de nouvelles clauses apprises durant la recherche précédente. La plupart des solveurs effectuent un redémarrage après qu'un seuil de clauses apprises soit atteint. Ainsi, à chaque redémarrage, le parcours de l'espace de

recherche change car l'ensemble de clauses est modifié. Il faut noter que la plupart des solveurs effacent régulièrement les clauses apprises. Ainsi, le mixage des redémarrages et des oublis peut dans certains cas mettre en cause la complétude des solveurs. Un paramétrage des redémarrages est donc nécessaire pour garder cette dernière propriété. Le solveur Chaff, par exemple, augmente à chaque redémarrage le seuil au-delà duquel une clause est oubliée. Ainsi, de plus en plus de clauses sont gardées, ce qui assure la complétude.

2.3.5.6 Structures de données paresseuses

Dans les solveurs SAT modernes, 80% du temps de calcul est consommé par la propagation unitaire. Ainsi, des structures de données dites « paresseuses » ont été introduites dans le but d'optimiser la propagation unitaire.

2.4 Exercices

Exercice 31 (Résolvant) *Par résolution, nous avons :*

$$\frac{a+b \quad \overline{a}+\overline{b}}{b+\overline{b}}$$

Montrer que : $b+\overline{b} \not\models (a+b).(\overline{a}+\overline{b})$.

□

Exercice 32 (Résolvant) *Nous rappelons que deux clauses sont égales si et seulement si elles ont les mêmes ensembles de littéraux.*

- *Les clauses* $p+q+\overline{r}+q+p+s+q+\overline{r}$ *et* $s+q+\overline{r}+p$ *sont-elles égales ?*
- *Les clauses* $p+q+r+p$ *et* $q+r+q+r+q+r$ *sont-elles égales ? L'une est-elle incluse dans l'autre ? L'une est-elle la conséquence de l'autre ? Sont-elles équivalentes ?*
- *Indiquer tous les résolvants des clauses* $a+\overline{b}+c$ *et* $a+b+\overline{c}$. *Ces résolvants sont-ils valides ?*

□

Exercice 33 (Preuve) *Les ensembles de formules suivants sont insatisfaisables.*

- $\{a, a \Rightarrow b, \overline{b}\}$.
- $\{a+b, \overline{a}+c, \overline{a}+\overline{d}, d+\overline{c}, \overline{b}+a\}$.
- $\{a+b+c, \overline{a}+b, \overline{b}+c, \overline{c}+a, \overline{a}+\overline{b}+\overline{c}\}$.

En donner une preuve par résolution.

□

Exercice 34 (Formalisation et résolution,*) *Remarquons que : « x à moins que y » se formalise en* $\neg(x \Leftrightarrow y)$. *Dans une maison hantée, les esprits se manifestent sous deux formes différentes, un* **chant obscène** *et un* **rire sardonique**, *cependant nous pouvons influencer le comportement en jouant de l'orgue ou en brûlant de l'encens. Compte-tenu des données suivantes :*

(i) Le chant ne se fait pas entendre, à moins que nous jouions de l'orgue sans que le rire ne se fasse entendre.
(ii) Si nous brûlons de l'encens, le rire se fait entendre si et seulement si le chant se fait entendre.
(iii) (En ce moment) Le chant se fait entendre et le rire est silencieux.
Et de la conclusion :
(iv) (En ce moment) Nous jouons de l'orgue et ne brûlons pas d'encens.
Nous posons :

- *c : le chant se fait entendre.*
- *o : nous jouons de l'orgue.*
- *r : le rire se fait entendre.*
- *e : nous brûlons de l'encens.*

1. *Simplifier en produit de clauses* $\neg(x \Leftrightarrow y)$.
2. *Formaliser sous forme de produit de clauses les hypothèses et la négation de la conclusion.*
3. *Prouver par résolution que le raisonnement est correct.*

Autrement dit transformer le produit des hypothèses et de la négation de la conclusion en un produit de clauses, et en déduire la clause vide. □

Exercice 35 (Preuve,*) *Montrer, à l'aide d'une preuve par résolution, la correction du raisonnement suivant :*

$$r+q \Rightarrow t, t.q \Rightarrow r, q \models t \Leftrightarrow r.$$

□

Exercice 36 (Formalisation et preuve,*) *Montrer par résolution que le raisonnement suivant est contradictoire :*

- *Il fait beau à moins qu'il neige.*
- *Il pleut à moins qu'il neige.*
- *Il fait beau à moins qu'il pleuve.*

□

Exercice 37 (Définir une clause) *Une clause est la clause vide ou une disjonction (non vide) de littéraux. Donner une définition formelle de ce qu'est une clause et définir par récurrence la fonction s telle que $s(C)$ est l'ensemble des littéraux de la clause C.*

□

Exercice 38 (Preuve) *Prouver la propriété 2.1.14 page 40 sur la monotonie et composition.* □

Exercice 39 (Propriété de la résolution) *Montrer la propriété 2.1.8 page 39.*

□

Exercice 40 (La résolution n'ajoute aucun littéral,*) *Soit Γ un ensemble de clauses. Un littéral de Γ est un littéral d'une clause de Γ. Montrer que toute clause déduite de Γ ne comporte que des littéraux de Γ.* □

Exercice 41 (Formalisation et résolution) *Considérons les hypothèses suivantes :*

1. *Si Pierre rate son tournoi alors Pierre sera déprimé.*
2. *S'il fait beau alors Pierre ira à la piscine.*
3. *Si Pierre ne va pas à la piscine il sera déprimé.*
4. *À la piscine, Pierre ne s'entraîne pas.*
5. *Pierre ratera son tournoi s'il ne s'entraîne pas.*

Nous souhaitons démontrer que des hypothèses précédentes, on peut déduire la conclusion suivante :

— *Pierre sera déprimé.*

Vous procéderez comme suit :

— *Formaliser les hypothèses et la négation de la conclusion.*
— *Déduire de vos énoncés formels un ensemble de clauses équivalent.*
— *Prouver qu'il est correct de déduire la conclusion à partir des hypothèses en démontrant avec une preuve* **par résolution** *que l'ensemble de clauses est contradictoire.*

□

Exercice 42 (Formalisation et résolution)

Les Beatles étaient un groupe de rock qui s'est formé dans les années soixantes à Liverpool. Ce groupe était composé de quatres garçons : Ringo Starr, Paul MacCartney, John Lennon et Georges Harrison. À l'époque où s'est formé le groupe, il n'a pas été évident de décider qui allait jouer de quel instrument. Pour preuve, voici un extrait de leurs discussions :

Paul dit : *« Si Ringo ne joue pas de la guitare, alors je jouerai de la basse et John jouera de la guitare »,*

Georges dit : *« Je jouerai de la guitare si et seulement si John en joue »,*

John dit : *« Si Paul joue de la basse, alors Georges jouera de la guitare »,*

Ringo dit : *« Je jouerai de la batterie et donc pas de la guitare ».*

Après cette discussion, ils décidèrent que :

— *Ringo jouerait à la batterie,*
— *Paul jouerait de la basse, et que*
— *John et Georges joueraient tous les deux de la guitare.*

Nous allons maintenant montrer que cette conclusion a satisfait tous les membres du groupe.

1. *Formalisez les quatres hypothèses et la conclusion en utilisant les variables propositionnelles suivantes :*
   — *RB : « Ringo joue de la batterie »,*
   — *RG : « Ringo joue de la guitare »,*
   — *PB : « Paul joue de la basse »,*
   — *JG : « John joue de la guitare », et*
   — *GG : « Georges joue de la guitare ».*
2. *Transformez en clauses les hypothèses et la négation de la conclusion.*
3. *Démontrez avec une preuve par résolution que la conclusion est conséquence des hypothèses.*

□

Exercice 43 (Réduction, X1603 Andrews) *Soit l'ensemble de clauses :*

$$\{p+q, \overline{p}+r+\overline{q}+p, p+\overline{r}, q+\overline{p}+\overline{q}, q+\overline{r}+p, r+q+\overline{p}+\overline{r}, \overline{r}+q\}.$$

1. *Réduire cet ensemble (la réduction est définie en 2.1.4 page 42)*
2. *Indiquer si l'ensemble réduit est ou non satisfaisable.*

□

Exercice 44 (DPLL) *Considérons l'ensemble de clauses suivant :*

$$\{\overline{a}+\overline{b}+\overline{f}, a+b+f, e+\overline{a}, \overline{a}+\overline{b}, \overline{a}+c, d+a+\overline{d}, a+b, \overline{a}+\overline{c}+\overline{d}, d\}.$$

— *Appliquer l'algorithme Algo_DPLL sur cet ensemble de clauses.*
— *Conclure si cet ensemble est satisfaisable ou non.*
— *Donner un modèle ou un contre-modèle obtenu à partir de la trace de l'algorithme.*

Dans l'arbre d'appel vous étiquetterez les étapes comme suit :
— *Suppression des clauses valides, en abrégé VAL.*
— *Réduction, en abrégé RE.*
— *Suppression des clauses ayant des littéraux isolés, en abrégé ELI.*
— *Résolution unitaire, abrégé en RU.*

De plus, noter les affectations effectuées à chaque étape de l'algorithme afin de retrouver facilement le modèle construit.

□

Exercice 45 (DPLL) *Utiliser l'algorithme Algo_DPLL pour déterminer si les ensembles suivants de clauses sont satisfaisables ou insatisfaisables :*
— $\{a+b+c+d+e+f, \overline{a}+b, \overline{b}+a, \overline{c}+d, \overline{d}+c, \overline{b}+\overline{c}, \overline{b}+c, b+\overline{c}, \overline{e}, \overline{f}\}$.
— $\{a+b+c+d+f, \overline{a}+b, \overline{b}+a, \overline{c}+d, \overline{d}+c\}$.
— $\{b+j+\overline{a}, a+j+\overline{b}, b+a+j, a+j, j+b, \overline{b}+\overline{j}, \overline{j}+b, j+s, \overline{s}+\overline{b}\}$.
— $\{a+\overline{c}+d, \overline{b}+c+f, b+\overline{e}+f, \overline{c}+e+\overline{f}, e+f, c+d, \overline{a}, \overline{e}+\overline{f}\}$.

Donner une trace de l'algorithme.

□

Exercice 46 (Stratégie complète) *Soit Γ le produit suivant de clauses :*

$$(a+b+\overline{c}).(b+\overline{c}).(c+\overline{c}).(b+c).(\overline{a}+\overline{b}+c).(a+\overline{b}+c).$$

Déterminer par la stratégie complète de résolution si Γ est insatisfaisable ou possède un modèle. Donner la trace de l'algorithme. Indiquer la ou les preuves obtenues. Si Γ a un modèle indiquez-le. □

Exercice 47 (Stratégie complète) *Soient l'ensemble de clauses suivant :*

$$\{p+q, \overline{p}+s, \overline{s}+t, \overline{t}, \overline{q}+r, \overline{r}, \overline{r}+p+t, q+z+\overline{z}, \overline{q}+r+s\}.$$

Appliquer l'algorithme de la stratégie complète sur cet ensemble de clauses et conclure si cet ensemble est satisfaisable ou non.

□

Exercice 48 (Stratégie complète) *Considérons la fonction f telle que $f(x,y,z)=0$ si et seulement si le nombre d'arguments valant 1 est pair. Exprimer f comme un produit de clauses suivant la méthode décrite dans la sous-section 1.6 page 29 puis simplifier f en utilisant la stratégie complète.*

□

Exercice 49 (Preuve) *Prouver le lemme 2.3.2 page 46.* □

Exercice 50 (Preuve) *Prouver le lemme 2.3.5 page 46.* □

Exercice 51 (De SAT à 3-SAT,*)** *SAT est un problème de décision qui peut être énoncé comme suit : « étant donné un ensemble de clauses Γ, existe-t-il une assignation qui soit modèle de Γ ? » 3-SAT est une restriction de ce problème où toutes les clauses de Γ ont exactement trois littéraux.*

SAT est un problème NP-complet [5]. Dans cet exercice, nous proposons d'étudier une réduction polynomiale de SAT vers 3-SAT, prouvant ainsi que 3-SAT est aussi NP-complet[2].

Une réduction polynomiale de SAT vers 3-SAT consiste à transformer en temps polynomial un ensemble de clauses Γ en un ensemble de clauses Γ' vérifiant les deux propriétés suivantes :

2. Il faut noter qu'en général, un problème k-SAT n'est pas nécessairement NP-complet. Par exemple, 2-SAT a été prouvé polynomial.

(a) Γ' *est uniquement constitué de clauses ayant trois littéraux distincts.*

(b) Γ *a un modèle si et seulement si* Γ' *a un modèle.*

Généralement, de telles réductions vérifient une propriété supplémentaire :

(c) Tout modèle de Γ' *est un modèle de* Γ.

Le but de cet exercice est de montrer que la transformation polynomiale présentée ci-dessous vérifie les trois propriétés énoncées ci-dessus. Noter que cette transformation introduit des variables supplémentaires.

Soit $\Gamma = \{c_1, \ldots, c_m\}$ *un ensemble de clauses. La réduction consiste à remplacer toute clause* $c_i = z_1 + \ldots + z_k$ *de* Γ *(ATTENTION :* $z_1, \ldots, z_k$ *sont des littéraux) par un ensemble de clauses* C'_i *construit en fonction de la valeur de k, comme suit :*

$k = 1$ **:** $C'_i = \{z_1 + y_1 + y_2,\ z_1 + y_1 + \overline{y_2},\ z_1 + \overline{y_1} + y_2,\ z_1 + \overline{y_1} + \overline{y_2}\}$ *où* y_1 *et* y_2 *sont des variables supplémentaires.*

$k = 2$ **:** $C'_i = \{z_1 + z_2 + y_1,\ z_1 + z_2 + \overline{y_1}\}$ *où* y_1 *est une variable supplémentaire.*

$k = 3$ **:** $C'_i = \{c_i\}$.

$k > 3$ **:** $C'_i = \{z_1 + z_2 + y_1,\ \overline{y_1} + z_3 + y_2,\ \overline{y_2} + z_4 + y_3,\ \overline{y_3} + z_5 + y_4,\ \ldots,\ \overline{y_{k-3}} + z_{k-1} + z_k\}$ *où* $y_1, \ldots, y_{k-3}$ *sont des variables supplémentaires.*

Ainsi, $\Gamma' = \bigcup_{i=1}^{m} C'_i$.

Par construction, la réduction proposée vérifie la propriété (a). *Les réponses aux questions suivantes permettent de prouver les propriétés* (b) *et* (c) *:*

1. *Montrer (sans table de vérité) que toute assignation donne la même valeur à* c_i *et* C'_i *lorsque* c_i *est constituée d'un seul littéral.*
2. *Montrer (sans table de vérité) que toute assignation donne la même valeur à* c_i *et* C'_i *lorsque* c_i *est constituée de deux littéraux.*
3. *Soit* c_i *une clause de plus de 3 littéraux. Montrer que si* c_i *a un modèle, alors* C'_i *a aussi un modèle.*
4. *Soit* c_i *une clause de plus de 3 littéraux. Montrer que tout modèle de* C'_i *est modèle de* c_i.
5. *La réduction proposée maintient uniquement la satisfaisabilité. Montrer qu'elle ne maintient pas le sens.*

□

Chapitre 3

Déduction Naturelle

Sommaire

3.1 Système formel de la déduction naturelle **58**
3.1.1 Règles de conjonction 58
3.1.2 Règles de disjonction 58
3.1.3 Règles de l'implication 59
3.1.4 Deux règles spéciales 59
3.1.5 Preuves en déduction naturelle 60
3.2 Tactiques de preuve **63**
3.3 Cohérence de la déduction naturelle **65**
3.4 Complétude de la déduction naturelle **66**
3.5 Outils **66**
3.5.1 Logiciel de construction automatique de preuves 66
3.5.2 Dessiner des arbres de preuves 66
3.6 Exercices **67**

UNE preuve dans les cours de Mathématiques est une décomposition du raisonnement en pas élémentaires évidents, nous pratiquons alors sans le savoir la *déduction naturelle*. Dans ce chapitre, nous présentons un système formel qui est une modélisation particulière de la déduction naturelle. La déduction naturelle et le calcul de séquent furent introduits en 1934 par Gerhard Gentzen (1909-1945) [12, 13]. Les preuves dans notre système ne sont pas des arbres comme dans le système original de Gehard Gentzen mais ressemblent plus aux preuves des mathématiciens. Nous expliquons aussi comment faire le lien entre ces deux formalismes en construisant des arbres proches de ceux introduits originellement. Nous choisissons de ne parler que de la *logique classique*, par opposition à la *logique intuitionniste* qui est obtenue en *omettant* la règle de réduction à l'absurde (la dernière règle de la table 3.1 page 59).

Remarque préliminaire : Pour simplifier l'étude de la déduction naturelle, nous considérons que le vrai, la négation et l'équivalence sont des *abréviations* ainsi définies :

— $\top$ abrège $\bot \Rightarrow \bot$.
— $\neg A$ abrège $A \Rightarrow \bot$.
— $A \Leftrightarrow B$ abrège $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

Deux formules seront considérées comme *égales*, si les formules obtenues en éliminant les abréviations sont identiques. Par exemple, les formules $\neg\neg a$, $\neg a \Rightarrow \bot$ et $(a \Rightarrow \bot) \Rightarrow \bot$ sont égales. Il est clair que deux formules égales, au sens ci-dessus, sont équivalentes (voir exercice 55 page 67) et nous utilisons implicitement cette propriété quand nous remplaçons une formule par une autre formule égale aux abréviations près.

Plan : Nous décrivons tout d'abord notre système de règles de déduction naturelle et introduisons la notion de preuve dans ce contexte. Ensuite, nous présentons des tactiques pour aider le lecteur à la rédaction de preuve. Nous montrons la cohérence et la complétude de notre système. Enfin nous présentons un outil permettant à partir d'une formule de construire automatiquement une preuve ou bien d'exhiber un contre-exemple.

3.1 Système formel de la déduction naturelle

Nous présentons d'abord les règles du système de la déduction naturelle. Elles constituent les raisonnements élémentaires autorisés. Ensuite nous exposons la notion de preuve qui se construit par applications successives de ces règles.

Définition 3.1.1 (Règle) *Une* règle *est constituée de formules dites hypothèses $H_1, \ldots, H_n$ et d'une conclusion. Les hypothèses sont écrites au-dessus d'un trait et l'unique formule en dessous de ce trait est la conclusion de la règle. Le nom d'une règle est écrit au même niveau que le trait séparant hypothèses et conclusion.*

$$\frac{H_1 \ldots H_n}{C}\ R$$

La déduction naturelle que nous présentons comporte une dizaine de règles de déduction qui sont regroupées dans la table 3.1 page suivante. La règle fondamentale est la suivante : si nous pouvons déduire une formule B d'une hypothèse A, alors nous pouvons déduire $A \Rightarrow B$ en nous passant de cette hypothèse. Nous décrivons chacune des dix règles du système de déduction que nous considérons. L'application de ces règles et seulement de ces règles va nous permettre de prouver la correction des raisonnements. Il est donc important de bien comprendre le fonctionnement de chacune d'entre elles. Nous avons deux familles de règles pour chaque connecteur : les règles d'introduction et les règles d'élimination. L'intuition du rôle des règles d'introduction est de générer un symbole de la logique afin de bâtir progressivement la formule à prouver. Les règles d'élimination servent à simplifier un raisonnement ou à obtenir une nouvelle formule grâce aux raisonnements déjà réalisés et ainsi faire disparaître un connecteur logique. En plus nous avons deux règles spéciales, que nous décrirons en dernier, une pour éliminer deux occurrences successives de la négation et une autre pour déduire d'importe quoi à partir du faux.

3.1.1 Règles de conjonction

La règle d'introduction de la conjonction, notée $\wedge$I, signifie simplement que si nous avons une preuve de A et une preuve de B, alors nous pouvons déduire une preuve de la proposition $A \wedge B$. Nous pouvons aussi dire que la preuve de la proposition $A \wedge B$ peut se décomposer en une preuve de A et une preuve de B.

$$\frac{A \quad B}{A \wedge B}\ \wedge I$$

Les règles d'élimination de la conjonction, notées $\wedge E1$ et $\wedge E2$, permettent de déduire à partir de la conjonction de deux formules $A \wedge B$ soit la formule A soit la formule B. Ainsi nous avons éliminé le connecteur de la conjonction.

$$\frac{A \wedge B}{A}\ \wedge E1 \qquad \frac{A \wedge B}{B}\ \wedge E2$$

3.1.2 Règles de disjonction

Les règles d'introduction de la disjonction sont les duales des règles de l'élimination de la conjonction. Les deux règles d'introduction, notées $\vee$I1 et $\vee$I2, permettent de créer une disjonction de deux formules à partir d'une des deux formules. Ces règles signifient simplement que si A est vraie alors $A \vee B$ et $B \vee A$ sont aussi vraies quelle que soit la proposition B.

$$\frac{A}{A \vee B}\ \vee I1 \qquad \frac{A}{B \vee A}\ \vee I2$$

La règle d'élimination de la disjonction, notée $\vee E$, est plus complexe. Afin de déduire la formule C à partir de la disjonction $A \vee B$ il faut aussi prouver les prémisses suivantes : A implique C et B implique C. Cette règle formalise la notion de *preuve par cas* utilisée en mathématiques : supposons que dans un environnement donné, nous souhaitons prouver C alors que deux cas, A ou B, sont possibles ; nous nous plaçons alors dans le cas où A est vérifiée et nous prouvons C, puis nous nous plaçons dans le cas où B est vérifiée et nous prouvons C.

$$\frac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C}\ \vee E$$

3.1.3 Règles de l'implication

La règle de l'élimination de l'implication $\Rightarrow E$ dit que si nous sommes capables d'obtenir une preuve de A et une preuve de $A \Rightarrow B$ alors nous avons une preuve de B. Cette règle correspond au *modus ponens*.

$$\frac{A \quad A \Rightarrow B}{B} \Rightarrow E$$

La règle fondamentale de notre système est la règle d'*introduction de l'implication* $\Rightarrow I$: si nous pouvons déduire une formule B d'une hypothèse A, alors nous pouvons déduire $A \Rightarrow B$ en nous passant de cette hypothèse. Cette règle est résumée par le schéma ci-dessous, où la notation $[A]$ indique que si A est une hypothèse de la preuve de B, cette hypothèse est *enlevée* de la preuve de $A \Rightarrow B$, autrement dit ne sert plus pour prouver $A \Rightarrow B$.

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B} \Rightarrow I$$

3.1.4 Deux règles spéciales

Enfin nous avons deux règles spéciales :

— La première permet de déduire n'importe quoi à partir du faux : il s'agit de la « règle du faux », notée Efq, pour la formule latine « ex falso, quodlibet », indiquant que du faux, on peut déduire ce qu'on veut.

$$\frac{\bot}{A} Efq$$

— La seconde est la règle de réduction à l'absurde, notée *RAA* pour « reductio ad absurdum ». Elle élimine deux occurrences successives de la négation.

$$\frac{\neg\neg A}{A} RAA$$

Nous regroupons l'ensemble des règles de la déduction naturelle dans la table 3.1.

Introduction	Élimination
Règle du faux	
Règle de réduction à l'absurde	

TABLE 3.1 – Ensemble de règles de déduction naturelle.

3.1.5 Preuves en déduction naturelle

Intuitivement une preuve est l'application successive de règles de la déduction naturelle. Dans le système originel introduit par Gerhard Gentzen, une preuve est représentée par un arbre de formules (certaines formules pouvant être marquées comme enlevées). Nous avons choisi de représenter une preuve par une succession d'étapes de raisonnement basées sur les règles de la déduction naturelle, ceci afin d'être plus proche des preuves faites en mathématiques. Dans le chapitre précédent, une preuve par résolution était constituée d'une liste de clauses. En déduction naturelle, au cours d'une preuve, nous pouvons ajouter des hypothèses et les enlever, ce qui rend plus complexe la définition de ce qu'est une preuve. Nous introduisons donc plusieurs notions intermédiaires afin de définir clairement la notion de preuve en déduction naturelle. Tout d'abord nous introduisons les composants d'une preuve, c'est-à-dire, les « lignes d'une preuve », puis la notion de brouillon de preuve, ensuite les contextes de ces lignes de preuve afin de définir une preuve en déduction naturelle.

3.1.5.1 Brouillon de preuve

Une preuve est constituée d'une succession de lignes qui sont soit une formule soit une formule précédée par le mot clef « `Supposons` » ou « `Donc` ».

Définition 3.1.2 (Ligne de preuve) *Une* ligne *de preuve est de l'une des trois formes suivantes :*
- `Supposons` formule,
- formule,
- `Donc` formule.

Un brouillon de preuve est une preuve en construction, intuitivement elle a au moins autant de `Supposons` que de `Donc`.

Définition 3.1.3 (Brouillon de preuve) *Un* brouillon *de preuve est une suite de lignes telle que, dans tout préfixe de la suite, le nombre de lignes commençant par le mot* `Supposons` *est au moins égal à celui des lignes commençant par le mot* `Donc`.

Exemple 3.1.4 *Dans l'exemple ci-dessous, la suite des lignes de 1 à 3 est un brouillon de preuve. Par contre la suite des lignes de 1 à 5,* n'est pas un brouillon de preuve *car dans la suite des lignes 1 à 4, le nombre de lignes commençant par le mot* `Donc` *dépasse celui des lignes commençant par le mot* `Supposons`.

numéro	*ligne*
1	`Supposons` a
2	$a \vee b$
3	`Donc` $a \Rightarrow a \vee b$
4	`Donc` $\neg a$
5	`Supposons` b

3.1.5.2 Contexte des lignes d'un brouillon de preuve

Afin de pouvoir appliquer la règle d'introduction de l'implication nous avons besoin de connaître à chaque étape d'une preuve les formules qui sont des hypothèses utilisables. Pour ce faire, nous associons à chaque ligne d'un brouillon de preuve un *contexte*, qui est la suite des hypothèses introduites par les lignes `Supposons` et non enlevées par les lignes `Donc`. Puisque dans tout préfixe d'un brouillon de preuve, le nombre de lignes commençant par le mot `Supposons` est au moins égal à celui des lignes commençant par le mot `Donc`, le contexte de la ligne précédant une ligne `Donc` n'est pas vide et par suite le contexte de chaque ligne d'un brouillon de preuve est bien défini. Dans la suite, nous représentons les contextes de chaque ligne en remplaçant chaque hypothèse du contexte par le numéro de la ligne où cette hypothèse est introduite.

Définition 3.1.5 (Contexte) *Nous numérotons les lignes d'un brouillon de preuve de 1 à n. Pour i de 1 à n, la liste de formules* Γ_i *est le* contexte *de la ligne i. La liste* Γ_0 *est vide et les listes de formules* Γ_i *sont ainsi définies :*
- *Si la ligne i est « `Supposons` A », alors* $\Gamma_i = \Gamma_{i-1}, i$.
- *Si la ligne i est « A » alors* $\Gamma_i = \Gamma_{i-1}$
- *Si la ligne i est « `Donc` A » alors* Γ_i *est obtenue en enlevant la dernière formule de* Γ_{i-1}

La liste Γ_i *est le* contexte *de la ligne i.*

Nous présentons nos preuves sous forme de tableaux, ce qui nous permet de numéroter les lignes d'une preuve, les contextes utilisables, et les règles utilisées pour générer une nouvelle ligne.

Exemple 3.1.6 *Nous illustrons ici la gestion des contextes sur un brouillon de preuve simple.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` a	
1,2	*2*	`Supposons` b	
1,2	*3*	$a \wedge b$	$\wedge I$ *1,2*
1	*4*	`Donc` $b \Rightarrow a \wedge b$	$\Rightarrow I$ *2,3*
1,5	*5*	`Supposons` e	

3.1.5.3 Preuves

Nous précisons la notion de conclusion et de formule utilisable afin de définir et manipuler des preuves en déduction naturelle.

Définition 3.1.7 (Conclusion, formule utilisable) *Nous définissons la notion de conclusion et de formule utilisable :*
- *La formule figurant sur une ligne d'un brouillon de preuve est la* conclusion *de la ligne.*
- *La conclusion d'une ligne est* utilisable *tant que son contexte (c'est-à-dire les hypothèses qui ont permis de la déduire) est présent.*

Autrement dit la conclusion de la ligne i est utilisable sur la ligne i et sur toutes les lignes suivantes dont le contexte a pour préfixe le contexte de la ligne i.

Exemple 3.1.8 *Dans l'exemple ci-dessous, la conclusion de la ligne 2 est utilisable sur la ligne 2, et pas au delà, car à la ligne 3, l'hypothèse qui a permis de la déduire est enlevée.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` a	
1	*2*	$a \vee b$	$\vee I1$ *1*
	3	`Donc` $a \Rightarrow a \vee b$	$\Rightarrow I$ *1,2*
4	*4*	`Supposons` c	
4	*5*	$c \vee a$	$\vee I$ *4*
	6	`Donc` $c \Rightarrow c \vee a$	$\Rightarrow I$ *4,5*
	7	$(a \Rightarrow a \vee b) \wedge (c \Rightarrow c \vee a)$	$\wedge I$ *3,6*

Nous donnons la définition de preuve dans un environnement. Un environnement est un ensemble de formules supposées « vraies ». Intuitivement, réaliser une preuve dans un environnement consiste à pouvoir utiliser sans les prouver les formules de l'environnement.

Définition 3.1.9 (Preuve) *Soit* Γ *un ensemble de formules,* une preuve dans l'environnement Γ *est un brouillon de preuve ayant les propriétés suivantes :*

1. *Pour toute ligne « `Donc` A », la formule A est égale à* $B \Rightarrow C$*, où B est la dernière formule du contexte de la ligne précédente et où C est une formule utilisable sur la ligne précédente ou égale à un élément de l'environnement* Γ.
2. *Pour toute ligne « A », la formule A est la conclusion d'une règle (autre que la règle d'introduction de l'implication) dont les prémisses sont utilisables sur la ligne précédente ou sont éléments de l'environnement* Γ.

La ligne « `Donc` A » est une application de la règle d'introduction de l'implication. En effet, C est déduite de Γ ou d'hypothèses qui figurent dans la ligne précédente. La liste des hypothèses de la ligne précédente se terminant par B, nous pouvons en déduire $B \Rightarrow C$, en nous passant de l'hypothèse B.

Définition 3.1.10 (Preuve d'une formule) *Une preuve de la formule A dans l'environnement* Γ *est soit la preuve vide lorsque A est élément de* Γ*, soit une preuve dont la dernière ligne est de conclusion A et de contexte vide.*

Nous notons $\Gamma \vdash A$ le fait qu'il y a une preuve de A dans l'environnement Γ et $\Gamma \vdash P : A$ le fait que P est une preuve de A dans l'environnement Γ. Lorsque l'environnement est vide, nous l'omettons, autrement dit nous abrégeons $\emptyset \vdash A$ en $\vdash A$. Lorsque nous demandons une preuve d'une formule sans indiquer d'environnement, nous supposons que l'environnement est vide.

Exemple 3.1.11 *Prouvons* $(a \Rightarrow b) \Rightarrow (\neg b \Rightarrow \neg a)$.

contexte	*numéro*	*preuve*	*règle*
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	8		

La preuve elle-même est ce qui figure dans la colonne preuve. Nous y avons ajouté la numérotation des lignes de preuves, les justifications qui indiquent les règles utilisées et les contextes de chaque ligne de la preuve.

Nous présentons la même preuve sous forme d'arbre, ce qui correspond à l'approche originelle de Gerhard Gentzen.

Nous remarquons que pour lever une hypothèse il est d'usage de barrer cette hypothèse.

Exemple 3.1.12 *Nous donnons la preuve de* $\neg A \vee B$ *dans l'environnement* $A \Rightarrow B$. *Nous suggérons de numéroter par i, ii, iii, etc... les formules de l'environnement pour distinguer ces numéros et les numéros des lignes de la preuve.*

environnement			
référence		*formule*	
i		$A \Rightarrow B$	
contexte	*numéro*	*preuve*	*règle*
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		

Nous présentons aussi la preuve précédente sous forme d'arbre.

Comme il a été rappelé dans les remarques préliminaires, nous considérons comme égales deux formules qui sont identiques quand nous éliminons leurs abréviations. Ainsi dans l'exemple précédent, nous avons identifié $A \Rightarrow \bot$ et $\neg A$ ainsi que $(\neg A \vee B) \Rightarrow \bot$ et $\neg(\neg A \vee B)$.

3.2 Tactiques de preuve

Nous donnons des conseils pour construire des preuves. Ces conseils sont la base du logiciel présenté dans le paragraphe 3.5 page 66. Pour prouver la formule A dans un environnement Γ, nous appliquons *dans l'ordre* les règles suivantes, où l'ordre a été choisi de façon à reculer le plus possible l'usage de la règle *RAA* :

1. Si $A \in \Gamma$, alors la preuve obtenue est vide.
2. Si A est la conséquence d'une règle dont les premisses sont dans Γ, alors la preuve obtenue est « A ».
3. Si Γ comporte une contradiction, c'est-à-dire une formule B et une formule $\neg B$, alors la preuve obtenue est « $\bot$, A ».
4. Si $A = B \wedge C$, alors :
   — prouver B : Soit P la preuve obtenue pour B,
   — prouver C : Soit Q la preuve obtenue pour C.
   La preuve obtenue pour A est « P, Q, A ».

   Les preuves peuvent échouer (si l'on demande de prouver une formule improuvable dans l'environnement donné) : Si la preuve de B ou C échoue, il en est de même de la preuve de A. Pour simplifier la suite, nous ne signalons plus les cas d'échecs, sauf s'ils doivent être suivis d'une autre preuve.
5. Si $A = B \Rightarrow C$, alors prouver C sous l'hypothèse B (qui est ajoutée à l'environnement). Soit P, la preuve obtenue pour C, la preuve obtenue pour A est « `Supposons` B, P, `Donc` A ».
6. Si $A = B \vee C$, alors prouver B : Si P est la preuve obtenue pour B, alors « P, A » est la preuve obtenue pour A. Si la preuve de B échoue, alors prouver C : Si P est la preuve obtenue pour C, alors « P, A » est la preuve obtenue pour A. *Si la preuve de C échoue, essayer les règles suivantes.*
7. Si $B \wedge C$ est dans l'environnement, alors prouver A à partir des formules B, C, qui remplacent $B \wedge C$ dans l'environnement et soit P le résultat de cette preuve. Alors « B, C, P » est une preuve de A dans l'environnement initial.
8. Si $B \vee C$ est dans l'environnement, alors :
   — prouver A dans l'environnement où B remplace $B \vee C$: Soit P la preuve obtenue,
   — prouver A dans l'environnement où C remplace $B \vee C$: Soit Q la preuve obtenue.
   La preuve de A est alors « `Supposons` B, P, `Donc` $B \Rightarrow A$, `Supposons` C, Q, `Donc` $C \Rightarrow A$, A ».
9. Si $\neg(B \vee C)$ est dans l'environnement, alors nous déduisons $\neg B$ par la preuve $P4$ et $\neg C$ par la preuve $P5$ (preuves demandées à l'exercice 59 page 68). Soit P la preuve de A dans l'environnement où $\neg B$, $\neg C$ remplacent la formule $\neg(B \vee C)$. La preuve de A est « $P4$, $P5$, P ».

10. Si $A = B \vee C$, alors prouver C sous l'hypothèse $\neg B$: soit P la preuve obtenue. « `Supposons` $\neg B$, P, `Donc` $\neg B \Rightarrow C$ » est une preuve de la formule $\neg B \Rightarrow C$ qui est équivalente à A. Pour obtenir la preuve de A, il suffit d'ajouter la preuve $P1$, demandé à l'exercice 59 page 68, de A dans l'environnement $\neg B \Rightarrow C$. La preuve obtenue de A est donc « `Supposons` $\neg B$, P, `Donc` $\neg B \Rightarrow C$, $P1$ ».
11. Si $\neg(B \wedge C)$ est dans l'environnement, alors nous en déduisons $\neg B \vee \neg C$ par la preuve $P3$ demandée à l'exercice 59 page 68 puis nous raisonnons par cas comme ci-dessous :
    — prouver A dans l'environnement où $\neg B$ remplace $\neg(B \wedge C)$: Soit P la preuve obtenue,
    — prouver A dans l'environnement où $\neg C$ remplace $\neg(B \wedge C)$: Soit Q la preuve obtenue.
    La preuve de A est « $P3$, `Supposons` $\neg B$, P, `Donc` $\neg B \Rightarrow A$, `Supposons` $\neg C$, Q, `Donc` $\neg C \Rightarrow A$, A ».
12. Si $\neg(B \Rightarrow C)$ est dans l'environnement, alors nous déduisons B par la preuve $P6$, $\neg C$ par la preuve $P7$ (preuves demandées à l'exercice 59 page 68). Soit P la preuve de A dans l'environnement où B, $\neg C$ remplacent la formule $\neg(B \Rightarrow C)$. La preuve de A est « $P6$, $P7$, P ».
13. Si $B \Rightarrow C$ est dans l'environnement et si $C \neq \bot$, autrement dit si $B \Rightarrow C$ n'est pas égale à $\neg B$, alors, nous déduisons $\neg B \vee C$ dans l'environnement $B \Rightarrow C$ par la preuve $P2$ demandée à l'exercice 59 page 68 puis nous raisonnons par cas :
    — prouver A dans l'environnement où $\neg B$ remplace $B \Rightarrow C$: Soit P la preuve obtenue,
    — prouver A dans l'environnement où C remplace $B \Rightarrow C$: Soit Q la preuve obtenue.
    La preuve de A est « $P2$, `Supposons` $\neg B$, P, `Donc` $\neg B \Rightarrow A$, `Supposons` C, Q, `Donc` $C \Rightarrow A$, A ».

Exemple 3.2.1 *Nous appliquons ces tactiques à la preuve de la loi de Peirce*[1] *:* $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$. *Cette formule n'est pas prouvable en logique intuitionniste, qui est, ainsi qu'il est dit au début du chapitre, la logique classique privée de la règle de réduction à l'absurde. Nous pouvons montrer, qu'en logique intuitionniste, cette formule est équivalente à la loi du tiers-exclu.*

Vu la forme de la formule, nous devons appliquer la tactique 5. La preuve de la formule de Peirce est donc de la forme suivante :

Preuve Q :
`Supposons` $(p \Rightarrow q) \Rightarrow p$
Q_1 *preuve de* p *dans l'environnement* $(p \Rightarrow q) \Rightarrow p$
`Donc` $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$

La preuve Q_1 utilise nécessairement la tactique 13. Donc cette preuve s'écrit : Dans l'environnement $B \Rightarrow C$ où $B = p \Rightarrow q$, $C = p$.

Preuve Q_1 *:*
$Q_{11} = P2$ *où* $P2$ *est la preuve de* $\neg B \vee C$ *dans l'environnement* $B \Rightarrow C$, *voir exercice 59 page 68*
`Supposons` $\neg B$
Q_{12} *preuve de* $A = p$ *dans l'environnement* $\neg B$
`Donc` $\neg B \Rightarrow A$
`Supposons` C
Q_{13} *preuve de* $A = p$ *dans l'environnement* C
`Donc` $C \Rightarrow A$
A

Q_{13} est la preuve vide, car $A = C = p$.
Q_{12} est la preuve de $A = p$ dans l'environnement $\neg(p \Rightarrow q)$. Cette preuve est la preuve $P6$ demandée à l'exercice 59 page 68, où $B = p$ et $C = q$. En recollant les morceaux Q_1, Q_{11}, Q_{12}, Q_{13}, nous obtenons la preuve Q, que nous ne recopions pas ici, car elle peut être obtenue automatiquement par le logiciel développé par Michel Levy, disponible à l'adresse suivante : `http://teachinglogic.univ-grenoble-alpes.fr/DN/`

Nous montrons ci-dessous comme retrouver la preuve Q_{12} sans utiliser les tactiques. La seule règle, ne conduisant pas à une impasse, est la réduction à l'absurde. Donc cette preuve est de la forme :

1. Charles Sanders Peirce était un logicien et philosophe américain (1839 - 1914).

Preuve Q_{12} de p dans l'environnement $\neg(p \Rightarrow q)$
`Supposons` $\neg p$
$\boxed{Q_{121} \text{ preuve de } \bot \text{ dans l'environnement } \neg(p \Rightarrow q),\ \neg p}$
`Donc` $\neg\neg p$
p

Pour obtenir une contradiction, donc une preuve de $\bot$, il faut déduire $p \Rightarrow q$. Donc la preuve Q_{121} est :

`Supposons` p
$\bot$
q
`Donc` $p \Rightarrow q$
$\bot$

3.3 Cohérence de la déduction naturelle

La preuve de cohérence de la déduction naturelle consiste à montrer que chaque preuve construite à partir d'un ensemble de formules Γ donne une formule qui est déductible de Γ.

Théorème 3.3.1 (Cohérence de la déduction) *Si une formule A est déduite d'un environnement de formules Γ ($\Gamma \vdash A$), alors A est une conséquence de Γ ($\Gamma \models A$).*

Preuve : Soit Γ un ensemble de formules. Soit P une preuve de A dans cet environnement. Soient C_i la conclusion et H_i le contexte de la $i^{\text{ème}}$ ligne de la preuve P. Nous posons $H_0 = \emptyset$, la liste vide. Si la preuve P est vide, nous posons $C_0 = A$. Notons par $\Gamma,\ H_i$ l'ensemble des formules de l'ensemble Γ et de la liste H_i. Nous montrons par induction que pour tout k nous avons $\Gamma,\ H_k \models C_k$, ce qui implique que pour la dernière ligne n de cette preuve, comme par définition H_n est la liste vide et $C_n = A$, nous obtenons $\Gamma \models A$.

Cas de base : Supposons que A est déduite de Γ par la preuve vide. Alors A est élément de Γ, donc $\Gamma \models A$. Puisque $H_0 = \emptyset$, nous pouvons conclure : $\Gamma, H_0 \models A$, donc $\Gamma, H_0 \models C_0$.

Induction : Supposons que pour toute ligne $i < k$ de la preuve nous avons $\Gamma,\ H_i \models C_i$. Montrons que $\Gamma,\ H_k \models C_k$. Supposons que la ligne k est :

1. « `Supposons` C_k ». La formule C_k est la dernière formule de H_k, donc $\Gamma,\ H_k \models C_k$.
2. « `Donc` C_k ». La formule C_k est égale à la formule $B \Rightarrow D$, et B est la dernière formule de H_{k-1}. Nous distinguons deux cas, soit D est élément de Γ, soit D est utilisable sur la ligne précédente.
   (a) Dans le premier cas, puisque D est égale à une formule de Γ, D est conséquence de Γ donc de $\Gamma,\ H_k$. Puisque $B \Rightarrow D$ est conséquence de D, il en résulte que $\Gamma,\ H_k \models C_k$
   (b) Dans le deuxième cas, D est utilisable sur la ligne précédente. Donc il existe $i < k$ tel que $D = C_i$ et H_i est préfixe de H_{k-1}. Par *hypothèse de récurrence*, $\Gamma,\ H_i \models D$. Puisque H_i est préfixe de H_{k-1}, nous avons $\Gamma,\ H_{k-1} \models D$. Puisque B est la dernière formule de H_{k-1}, nous avons $H_{k-1} = H_k,\ B$ et donc $\Gamma,\ H_k,\ B \models D$, ce qui implique $\Gamma,\ H_k \models B \Rightarrow D$. Enfin puisque C_k est égale à $B \Rightarrow D$ et que deux formules égales sont équivalentes, nous avons $\Gamma,\ H_k \models C_k$.
3. « C_k ». Cette formule est la conclusion d'une règle de la table 3.1 page 59, appliquée à ses prémisses utilisables à la ligne précédente ou aux éléments de Γ. Considérons le seul cas de la règle $\wedge I$, les autres cas étant analogues. La formule C_k est égale à $(D \wedge E)$ et les prémisses de la règle sont D et E. Puisque D et E sont éléments de Γ ou utilisables à la ligne précédente, comme dans le cas précédent, en utilisant l'hypothèse de récurrence, nous avons : $\Gamma,\ H_{k-1} \models D$ et $\Gamma,\ H_{k-1} \models E$. Puisque la ligne k ne change pas les hypothèses, nous avons $H_{k-1} = H_k$, donc $\Gamma,\ H_k \models D$ et $\Gamma,\ H_k \models E$. Puisque C_k est égale à $(D \wedge E)$, nous avons : $D,\ E \models C_k$. Par suite $\Gamma,\ H_k \models C_k$.

□

3.4 Complétude de la déduction naturelle

Nous prouvons la complétude des règles uniquement pour les formules avec les symboles logiques $\bot$, $\wedge$, $\vee$, $\Rightarrow$. La complétude pour les formules obtenues en ajoutant les symboles logiques $\top$, $\neg$ et $\Leftrightarrow$ résulte immédiatement du fait que ces symboles peuvent être considérés comme des abréviations.

Vu l'absence de négation, nous définissons un littéral comme étant une variable ou une implication entre une variable et $\bot$. Soit x une variable, x et $x \Rightarrow \bot$ (qui peut être abrégé en $\neg x$) sont des littéraux complémentaires. Dans la suite nous distinguons une liste de formules Γ et $s(\Gamma)$ l'ensemble des formules de la liste. Pour simplifier les notations, nous utilisons la virgule pour ajouter un élément en début ou en fin de liste ainsi que pour concaténer deux listes, que ces listes soit des listes de formules ou des preuves.

Théorème 3.4.1 *Soient Γ un ensemble fini de formules et A une formule, si $\Gamma \models A$ alors $\Gamma \vdash A$.*

Remarque 3.4.2 *Nous admettons ce thèorème dans ce polycopié. On peut en donner une preuve constructive, c'est-à-dire qu'elle donne un ensemble complet de tactiques pour construire des preuves d'une formule dans un environnement. Cependant ces tactiques peuvent donner des preuves longues. En particulier si nous devons prouver une formule $(B \vee C)$, il vaut mieux en général suivre les tactiques données au paragraphe 3.2 page 63, c'est-à-dire essayer de prouver B, puis essayer de prouver C et* seulement en cas d'échec*, utiliser la tactique donnée dans la preuve de complétude, qui « réduit » cette preuve à une preuve de C en ajoutant l'hypothèse $\neg B$.*

3.5 Outils

Nous indiquons deux logiciels pour se familiariser avec la déduction naturelle. Le premier construit automatiquement des preuves comme nous les avons présentées, le deuxième illustre sur quelques exemples de manière interactive comment dessiner des preuves sous forme d'arbres.

3.5.1 Logiciel de construction automatique de preuves

Pour produire automatiquement des preuves, nous recommandons d'utiliser le logiciel :

`http://teachinglogic.univ-grenoble-alpes.fr/DN/`

Ce logiciel permet grâce à une syntaxe intuitive de saisir une formule et de générer automatiquement :
— si la formule est (syntaxiquement) incorrecte, un message d'erreur (en rouge) est produit au dessus de la formule
— si la formule est prouvable, sa preuve (sans annotation) est générée par le logiciel.
— si la formule est correcte mais n'est pas prouvable, un contre-modèle est proposé au dessus de la formule.
Si la formule est prouvable, il est aussi possible d'obtenir une version de la preuve annotée.

3.5.2 Dessiner des arbres de preuves

Ceux qui affectionnent les preuves sous forme d'arbres peuvent utiliser le logiciel suivant développé Laurent Théry :

`http://www-sop.inria.fr/marelle/Laurent.Thery/peanoware/Nd.html`

ou sa version Android :

`https://play.google.com/store/apps/details?id=org.inria.peanoware`

Ce logiciel se présente comme un jeu, il propose des formules à prouver et permet d'appliquer les règles de la déduction naturelle, de manière interactive, pour en construire les arbres de preuve. Quand vous réussissez à construire la preuve d'une formule, vous voyez la photo de Dag Prawitz, un des principaux logiciens qui a étudié les propriétés de la déduction naturelle.

3.6 Exercices

Exercice 52 (Brouillon de preuve) *La suite de lignes suivante n'est pas un brouillon de preuve. Indiquer le numéro i de la première ligne telle que les lignes 1 à $i-1$ soient un brouillon de preuve et que les lignes 1 à i ne soient pas un brouillon de preuve. Préciser le contexte des lignes 1 à $i-1$.*

contexte	*numéro*	*preuve*
	1	`Supposons` a
	2	`Supposons` b
	3	c
	4	`Donc` d
	5	`Supposons` e
	6	f
	7	`Donc` g
	8	h
	9	i
	10	`Donc` j
	11	`Donc` k
	12	l

□

Exercice 53 (Preuves de formules avec les règles spéciales) *Donner une preuve pour les formules suivantes :*

1. $a \Rightarrow \neg\neg a$,
2. $\neg\neg a \Rightarrow a$,
3. $a \Leftrightarrow \neg\neg a$,
4. $a \vee \neg a$.

□

Exercice 54 (Preuves simples de formules) *Donner une preuve pour les formules suivantes :*

1. $a \Rightarrow c$ *dans l'environnement* $a \Rightarrow b,\ b \Rightarrow c$.
2. $(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$.
3. $(a \Rightarrow b) \Rightarrow ((b \Rightarrow c) \Rightarrow (a \Rightarrow c))$.

□

Exercice 55 (Abréviations) *Soient A une formule et $dpl(A)$ la formule obtenue en remplaçant dans A, le vrai, les négations et les équivalences (en forme abréviées) par leur définition. $dpl(A)$ est la formule obtenue en « dépliant » A, d'où le nom dpl choisi pour cette fonction de dépliage.*

— *Définir par récurrence la fonction dpl.*
— *Montrer que les formules A et $dpl(A)$ sont équivalentes.*
— *En déduire que deux formules, égales aux abréviations près, sont équivalentes.*

□

Exercice 56 (Preuves de formules) *Donner une preuve pour les formules suivantes :*

1. $a \Rightarrow (b \Rightarrow a)$.
2. $a \wedge b \Rightarrow a$.
3. $\neg a \Rightarrow (a \Rightarrow b)$.
4. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$.
5. $a \wedge b \Rightarrow b \wedge a$.
6. $a \vee b \Rightarrow b \vee a$.
7. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow (a \wedge b \Rightarrow c)$.
8. $(a \wedge b \Rightarrow c) \Rightarrow (a \Rightarrow (b \Rightarrow c))$.

9. $(a \Rightarrow b) \wedge (c \Rightarrow d) \Rightarrow (a \wedge c \Rightarrow b \wedge d)$.

□

Exercice 57 (À propos de l'implication) *Donner une preuve pour les formules suivantes :*

1. *(**)* $a \Rightarrow b$ *dans l'environnement* $\neg a \vee b$.
2. $(\neg b \Rightarrow \neg a) \Rightarrow (a \Rightarrow b)$.

□

Exercice 58 (Algèbre de Boole) *Donner une preuve pour les formules suivantes :*

1. $\neg(a \wedge \neg a)$.
2. $a \vee a \Rightarrow a$.
3. $a \wedge a \Rightarrow a$.
4. $a \wedge (b \vee c) \Rightarrow a \wedge b \vee a \wedge c$.
5. $a \wedge b \vee a \wedge c \Rightarrow a \wedge (b \vee c)$.
6. $a \vee b \wedge c \Rightarrow (a \vee b) \wedge (a \vee c)$.
7. $(a \vee b) \wedge (a \vee c) \Rightarrow a \vee b \wedge c$.

□

Exercice 59 (Preuves de formules avec environnement) *Donner une preuve des formules suivantes :*

1. *P1 :* $B \vee C$ *dans l'environnement* $\neg B \Rightarrow C$.
2. *P2 :* $\neg B \vee C$ *dans l'environnement* $B \Rightarrow C$.
3. *P3 :* $\neg B \vee \neg C$ *dans l'environnement* $\neg(B \wedge C)$.
4. *P4 :* $\neg B$ *dans l'environnement* $\neg(B \vee C)$.
5. *P5 :* $\neg C$ *dans le même environnement* $\neg(B \vee C)$.
6. *P6 : B dans l'environnement* $\neg(B \Rightarrow C)$.
7. *P7 :* $\neg C$ *dans le même environnement* $\neg(B \Rightarrow C)$.

□

Exercice 60 (Preuves de formules) *Donner une preuve en déduction naturelle des formules suivantes :*

1. $\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b)$.
2. $(\neg a \wedge \neg b) \Rightarrow \neg(a \vee b)$.
3. *(*)* $(\neg a \vee \neg b) \Rightarrow \neg(a \wedge b)$.
4. *(**)* $(a \vee b) \wedge (\neg a \vee b) \Rightarrow b$.
5. *(***)* $\neg(a \wedge b) \Rightarrow (\neg a \vee \neg b)$.
6. *(***)* $(a \vee b) \wedge (\neg b \vee c) \Rightarrow a \vee c$.

□

Exercice 61 (Partiel 2011) *Prouver les formules suivantes en utilisant la déduction naturelle sous forme de tableau :*

1. $\neg p \wedge \neg(\neg p \wedge q) \Rightarrow \neg q$.
2. $((p \vee q) \vee (p \vee r)) \wedge \neg p \Rightarrow q \vee r$.
3. *(*)* $\neg p \vee \neg(p \wedge q) \Rightarrow \neg q \vee \neg p$.

□

Exercice 62 (Partiel 2013) *Donner une preuve des formules suivantes en utilisant la déduction naturelle sous forme de tableau :*

1. $(p \vee q) \Rightarrow (\neg p \wedge \neg q) \Rightarrow r$.
2. $((p \Rightarrow q) \wedge (q \Rightarrow r)) \wedge \neg r \Rightarrow \neg p$.
3. $(p \Rightarrow q) \Rightarrow ((p \wedge q) \vee \neg p)$.

□

Exercice 63 (Quelques questions posées en examen)

1. *Démontrer la formule suivante par déduction naturelle :* $(p \vee q) \wedge (p \Rightarrow r) \Rightarrow q \vee r$
2. *Soient les hypothèses :*
   — *(H1) : Si Pierre est grand, alors Jean n'est pas le fils de Pierre*
   — *(H2) : Si Pierre n'est pas grand, alors Jean est le fils de Pierre*
   — *(H3) : Si Jean est le fils de Pierre alors Marie est la soeur de Jean*
   Formaliser et démontrer qu'on peut en déduire la conclusion (C) : « Marie est la soeur de Jean ou Pierre est grand ou les deux à la fois » par déduction naturelle.

□

Deuxième partie

Logique du premier ordre

Chapitre 4

Logique du premier ordre

Sommaire

4.1 Syntaxe **74**
4.1.1 Formules strictes 74
4.1.2 Formules à priorité 76
4.2 Être libre ou lié **77**
4.2.1 Occurrences libres et liées 77
4.2.2 Variables libres et liées 77
4.3 Sens des formules **78**
4.3.1 Déclaration de symbole 78
4.3.2 Signature 78
4.3.3 Interprétation 79
4.3.4 Sens des formules 80
4.3.5 Modèle, validité, conséquence, équivalence 82
4.3.6 Instanciation 82
4.3.7 Interprétation finie 83
4.3.8 Substitution et remplacement 86
4.4 Équivalences remarquables **86**
4.4.1 Relation entre $\forall$ et $\exists$ 86
4.4.2 Déplacement des quantificateurs 87
4.4.3 Changement de variables liées 87
4.5 Exercices **89**

Ce célèbre syllogisme[1] ne peut pas se formaliser en logique propositionnelle :

Tous les hommes sont mortels ;
or Socrate est un homme ;
donc Socrate est mortel.

Pour formaliser un tel raisonnement nous avons besoin d'enrichir la logique propositionnelle avec de nouveaux connecteurs appelés *quantificateurs*, cette logique étendue s'appelle logique du premier ordre. Elle permet de parler de structures comportant un seul domaine non vide (contrairement à la logique propositionelle, ce domaine peut avoir plus de deux valeurs), des fonctions et relations sur ce domaine. Le langage de cette logique comporte plusieurs catégories : les termes qui représentent les éléments du domaine ou des fonctions sur ces éléments, des relations qui relient des termes entre eux et les formules qui décrivent les interactions entre les relations grâce aux connecteurs et aux quantificateurs. Par exemple, la relation $mortel(x)$ désigne x est mortel, la relation $homme(x)$ signifie que x est un homme. De plus, $Socrate$ est une constante du domaine (c'est-à-dire, elle a pour valeur un élément du domaine) et $homme(Socrate)$ signifie que Socrate est un homme. Enfin, la formule $\forall x\ (homme(x) \Rightarrow mortel(x))$ indique que tous les hommes sont mortels. À partir de ces deux hypothèses, il est possible de conclure que Socrate est mortel ($mortel(Socrate)$) par différentes méthodes

1. La notion de syllogisme fut introduite par Aristote et signifie littéralement « parole (qui va) avec (une autre) », aussi connu en latin par *modus ponendo ponens* = « manière d'affirmer, d'établir en affirmant » et plus brièvement *modus ponens*.

que nous détaillerons dans cette seconde partie. Ce même raisonnement s'applique également pour prouver le syllogisme suivant :

Un cheval bon marché est rare.
Tout ce qui est rare est cher.
Donc un cheval bon marché est cher.

Malgré la conclusion qui semble contradictoire, le raisonnement est correct. Afin d'obtenir une contradiction il faut ajouter l'hypothèse suivante : $\forall x($ bonmarché $(x) \Leftrightarrow \neg$ cher $(x))$. Ceci traduit la relation communément admise entre la notion de bon marché et de cher, *i.e.*, tout ce qui est cher n'est pas bon marché et réciproquement. Avec cette hypothèse supplémentaire nous pouvons montrer que ce raisonnement est contradictoire, sans cette hypothèse le raisonnement est correct. Ceci élucide le paradoxe de ce syllogisme.

Notre objectif dans ce chapitre est d'introduire les concepts et notions élémentaires de la logique du premier ordre afin de pouvoir proposer des méthodes de raisonnement dans les chapitres suivants.

Plan : Nous commençons par décrire la syntaxe de la logique du premier ordre. Puis, nous définissons les notions de *libre* et *liée* qui sont essentielles pour pouvoir interpréter le sens des formules. Ensuite nous définissons le sens des formules que nous pouvons construire à partir de la syntaxe introduite précédemment. Enfin, nous énonçons des propriétés remarquables de la logique du premier ordre, qui pourront être utilisées dans les raisonnements.

4.1 Syntaxe

Nous ajoutons à la logique propositionnelle deux symboles : le symbole *existentiel* ($\exists$) et le symbole *universel* ($\forall$). Le symbole existentiel signifie qu'il existe un élément ayant une certaine propriété alors que le symbole universel permet de parler de tous les éléments ayant une propriété. Ces changements impliquent qu'un élément de ce langage peut désormais dépendre de plusieurs variables. Il faut donc rajouter dans la syntaxe un délimiteur de variables. Nous avons choisi classiquement la virgule. Ces changements introduisent de nouveaux symboles en plus des variables, ces symboles peuvent par exemple être des nombres sur lesquels nous pouvons créer des fonctions (comme l'addition) ou encore définir des relations (comme l'égalité). Toutes ces modifications rendent la présentation de la syntaxe légèrement plus subtile que celle de la logique propositionnelle.

4.1.1 Formules strictes

Pour écrire les formules de la logique du premier ordre nous étendons la syntaxe de la logique propositionnelle, ainsi nous disposons du vocabulaire suivant :

- Deux constantes propositionnelles : $\bot$ et $\top$ représentant respectivement le faux et le vrai.
- Variables : suite de lettres et de chiffres commençant par une des minuscules u, v, w, x, y, z.
- Connecteurs : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.
- Quantificateurs : $\forall$ le quantificateur universel et $\exists$ le quantificateur existentiel.
- Ponctuations : la virgule « , » et les parenthèses ouvrantes « (» et fermantes «) ».
- Symbole :
  - ordinaire : suite de lettres et de chiffres ne commençant pas par une des minuscules u, v, w, x, y, z.
  - spécial : $+, -, *, /, =, \neq, <, \leq, >, \geq$

L'ensemble des symboles spéciaux peut être augmenté de nouveaux éléments, suivant les domaines mathématiques étudiés, à condition de respecter la contrainte que ces nouveaux éléments ne soient pas dans les ensembles des constantes propositionnelles, des variables, des connecteurs, des quantificateurs, des ponctuations ou des symboles ordinaires.

Exemple 4.1.1 *Ci-dessous, nous illustrons ces notions :*

- $x, x1, x2, y$ *sont des variables,*
- *homme, parent, succ, 12, 24,* $f1$ *sont des symboles ordinaires, les symboles ordinaires représenteront des fonctions (constantes numériques ou fonctions à plusieurs arguments) ou des relations (variables propositionelles ou relations à plusieurs arguments).*
- $x = y$, $z > 3$ *sont des exemples d'application de symboles spéciaux.*

Nous définissons en toute généralité la notion de terme qui est essentielle en logique du premier ordre. Nous étendrons dans la suite cette notion en définissant ce qu'est un terme associé à un ensemble de symboles.

Définition 4.1.2 (Terme) *Un* terme *est défini de manière inductive par :*
- *un symbole ordinaire est un terme,*
- *une variable est un terme,*
- *si $t_1, \ldots, t_n$ sont des termes et si s est un symbole (ordinaire ou spécial) alors $s(t_1, \ldots, t_n)$ est un terme.*

Exemple 4.1.3

sont des termes, par contre

n'est pas un terme. Notons que $42(1, y, 3)$ est aussi un terme, mais l'usage veut que les noms des fonctions et relations soient des symboles ordinaires commençant par des lettres.

Définition 4.1.4 (Formule atomique) *Nous définissons une formule atomique de manière inductive par :*
- *$\top$ et $\bot$ sont des formules atomiques*
- *un symbole ordinaire est une formule atomique*
- *si $t_1, \ldots, t_n$ sont des termes et si s est un symbole (ordinaire ou spécial) alors $s(t_1, \ldots, t_n)$ est une formule atomique.*

Exemple 4.1.5

sont des formules atomiques, notons également que

ne sont pas des formules atomiques.

Notons que l'ensemble des termes et l'ensemble des formules atomiques ne sont pas disjoints. Par exemple, $p(x)$ est à la fois un terme et une formule atomique. Lorsque t est à la fois un terme et une formule atomique, nous distinguons $[\![t]\!]$ le sens de t vu comme un terme, de $[t]$ le sens de t vu comme une formule (voir le paragraphe 4.3.4 page 80).

Définition 4.1.6 (Formule) *Nous définissons une* formule *de manière inductive par :*
- *une formule atomique est une formule,*
- *si A est une formule alors $\neg A$ est une formule,*
- *si A et B sont des formules et si $\circ$ est une des opérations $\vee, \wedge, \Rightarrow, \Leftrightarrow$ alors $(A \circ B)$ est une formule ,*
- *si A est une formule et si x est une variable* quelconque *alors $\forall x\, A$ et $\exists x\, A$ sont des formules*[2].

Exemple 4.1.7

sont des formules atomiques, donc des formules. Par contre

est une formule qui n'est pas atomique.

La notion de *sous-formule* (définition 1.1.4) s'étend naturellement à la logique du premier ordre. La notion de *taille de formule stricte* nécessite qu'en à elle une nouvelle définition :

Définition 4.1.8 (Taille d'une formule) *La* taille d'une formule *A, notée $|A|$, est définie inductivement par :*
- *$|\top| = 0$ et $|\bot| = 0$.*
- *Si A est une formule atomique alors $|A| = 0$.*
- *$|\neg A| = 1 + |A|$.*
- *$|Qx\, A| = 1 + |A|$ si Q est un des quantificateurs $\forall$ ou $\exists$.*
- *$|(A \circ B)| = |A| + |B| + 1$.*

2. Attention, x peut ne pas exister dans A.

4.1.2 Formules à priorité

Nous reprenons les priorités de la logique propositionnelle pour les connecteurs, et nous ajoutons une priorité identique à celle de la négation pour les quantificateurs. Dans le tableau 4.1 nous donnons les priorités des symboles et des connecteurs par ordre décroissant du haut vers le bas. Nous indiquons seulement deux règles de formation des formules à priorité, qui les distinguent des formules complètement parenthésées et permettent d'omettre des parenthèses ou d'ajouter des nouvelles parenthèses.

Pour abréger l'écriture des termes, certains symboles de fonction $+,-,*,/$ et certains symboles de relations $=,\neq,<,\leq,>,\geq$ peuvent être écrits de manière infixe, c'est-à-dire, de façon usuelle.

Exemple 4.1.9 *Nous* abrégeons *le terme* $+(x,*(y,z))$ *en* $x+y*z$ *et* $\leq(*(3,x),+(y,5))$ *en* $3*x\leq y+5$. *La transformation inverse est définie en donnant aux symboles* $=,\neq,<,\leq,>,\geq$ *des priorités inférieures à celle des symboles* $+,-,*,/$.

OPÉRATIONS	
$-+$ unaire	
$*,/$	associatif gauche
$+,-$ binaire	associatif gauche
RELATIONS	
$=,\neq,<,\leq,>,\geq$	
NÉGATION, QUANTIFICATEURS	
$\neg,\forall,\exists$	
CONNECTEURS BINAIRES	
$\wedge$	associatif gauche
$\vee$	associatif gauche
$\Rightarrow$	associatif **droit**
$\Leftrightarrow$	associatif gauche

TABLE 4.1 – Priorités des connecteurs et symboles.

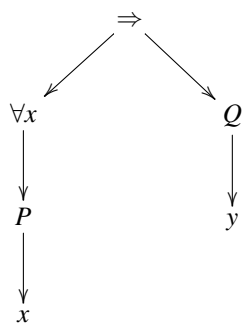
Définition 4.1.10 (Formule à priorité) *Une* formule à priorité *est :*

- *Une formule atomique.*
- *Si A est une formule à priorité alors* $\neg A$ *est une formule à priorité.*
- *Si A et B sont des formules à priorité alors* $A\circ B$ *est une formule à priorité.*
- *Si A est une formule à priorité et si x une variable* quelconque *alors* $\forall x\, A$ *et* $\exists x\, A$ *sont des formules à priorité.*
- *Si A est une formule à priorité alors* (A) *est une formule à priorité.*

Exemple 4.1.11 *La formule (à priorité)* $\forall xP(x)\wedge\forall xQ(x)\Leftrightarrow\forall x(P(x)\wedge Q(x))$ *peut être vue comme une* abréviation *de la formule* $((\forall xP(x)\wedge\forall xQ(x))\Leftrightarrow\forall x(P(x)\wedge Q(x)))$.

La formule (à priorité) $\forall x\forall y\forall z(x\leq y\wedge y\leq z\Rightarrow x\leq z)$ *peut être vue comme une* abréviation *de la formule* $\forall x\forall y\forall z((\leq(x,y)\wedge\leq(y,z))\Rightarrow\leq(x,z))$.

Exemple 4.1.12 *La priorité du* $\forall$ *est plus forte que celle de* $\Rightarrow$, *dans la formule* $\forall xP(x)\Rightarrow Q(y)$. *Donc, l'opérande gauche de l'implication est* $\forall xP(x)$. *La structure de la formule sera ainsi représentée par l'arbre suivant :*



De manière analogue à la logique propositionnelle, la taille d'une formule à priorité sera égale à la taille de la formule stricte dont elle est l'abréviation. De même, pour les sous-formules, nous considérerons toujours la formule stricte dont la formule à priorité est l'abréviation.

4.2 Être libre ou lié

Le sens de la formule $x+2=4$ dépend de x : la formule n'est vraie (en arithmétique) que si $x=2$. La variable x est libre dans cette formule.

Par contre, toujours en arithmétique $\forall x(x+2=4)$ est une formule fausse et $\exists x(x+2=4)$ est une formule vraie car $2+2=4$. Pour chacune de ces deux formules, il n'y a pas à choisir de valeur pour x afin de déterminer leur valeur respective : ces deux formules n'ont pas de variables libres.

4.2.1 Occurrences libres et liées

Sur une représentation en arbre, où nous dessinons les structures des formules en faisant apparaître $\forall x$ et $\exists x$ comme des sommets de l'arbre, une *occurrence liée* de la variable x est une occurrence en dessous d'un sommet $\exists x$ ou $\forall x$. Une occurrence de x qui n'est pas sous un tel sommet est *libre*.

Définition 4.2.1 (Portée de liaison, occurrence libre, liée) *Soient x une variable et A une formule. Dans une formule $\forall x\, A$ ou $\exists x\, A$, la* portée de la liaison *pour x est A. Une occurrence de x dans une formule est* libre *si elle n'est pas dans la portée d'une liaison pour x, sinon elle est dite* liée.

Exemple 4.2.2 *Pour voir les occurrences des variables, nous dessinons la structure de la formule $\forall x P(x,y) \wedge \exists z R(x,z)$, puisque la priorité de $\forall$ est supérieure à celle du $\wedge$, nous obtenons :*

L'occurrence de x présente dans $P(x,y)$ est liée, l'occurrence de x présente dans $R(x,z)$ est libre. L'occurrence de z est liée.

4.2.2 Variables libres et liées

Définition 4.2.3 (Variable libre, liée, formule fermée) *La variable x est une* variable libre *d'une formule si et seulement s'il y a une occurrence libre de x dans la formule. Une variable x est une* variable liée *d'une formule si et seulement s'il y a une occurrence liée de x dans la formule. Une formule sans variable libre est aussi appelée une* formule fermée.

Remarque 4.2.4 *Une variable peut-être à la fois libre et liée. Par exemple, dans la formule $\forall x P(x) \vee Q(x)$, x est à la fois libre et liée.*

Remarque 4.2.5 *Par définition, une variable qui n'apparaît pas dans une formule (0 occurrence) est une variable NON libre de la formule.*

Exemple 4.2.6 *Les variables libres de la formule de l'exemple 4.2.2 sont x et y.*

4.3 Sens des formules

Dans la suite, nous définissons en logique du premier ordre, le sens des connecteurs, des quantificateurs ainsi que de *l'égalité*. Aussi, pour définir le sens d'une formule, il suffit d'indiquer la valeur des variables libres et le sens de ses symboles utilisés. Dans un premier temps nous expliquons comment nous déclarons un symbole. Ensuite nous introduisons la notion de signature qui permet de définir les termes et formules associés. Une fois toutes ces notions établies nous définissons les interprétations afin de donner du sens aux formules de la logique du premier ordre.

4.3.1 Déclaration de symbole

Afin de donner un sens aux formules de la logique du premier ordre il nous faut déclarer les symboles utilisés. Nous distinguons en plus des variables, les symboles de fonctions, par exemple $g(x,y)$, et les symboles de relation comme $parent(x,y)$.

Définition 4.3.1 (Déclaration de symbole) *Une* déclaration de symbole *est un triplet noté s^{gn} où s est un symbole, g est soit f (pour fonction) ou r (pour relation), et n est un entier naturel dénotant le nombre d'arguments de ce symbole, n est aussi appelé l'*arité *de s.*

Exemple 4.3.2 *Les fonctions d'arité* 0 *constituent les constantes, par exemple* 1 *et* 2 *sont des constantes d'arité nulle, mais $parent^{r2}$ est une notation signifiant que parent est employé comme relation d'arité* 2*. De même $*^{f2}$ est une déclaration annonçant que $*$ est une fonction* 2*-aire. Le symbole A^{r0} est d'arité* 0 *et signifie que A est une variable propositionnelle. Le symbole $homme^{r1}$ est d'arité* 1 *ou* unaire *et le symbole g^{f2} est d'arité* 2 *ou* 2*-aire.*

Remarque 4.3.3 *Lorsque le contexte ou les conventions usuelles comportent une déclaration implicite d'un symbole, nous omettons l'exposant. Par exemple, le symbole égal étant toujours employé comme relation à deux arguments, nous abrégeons la déclaration de symbole $=^{r2}$ en $=$.*

4.3.2 Signature

En logique du premier ordre nous pouvons choisir le nom des variables utilisées mais nous avons aussi la possibilité de construire nos propres constantes (fonctions sans arguments), fonctions, variables propositionnelles (relations d'arité nulle) et relations. Une signature est donc l'ensemble des déclarations de symboles autorisés pour construire des formules. Elle permet de définir les symboles dont le sens n'est pas fixé *a priori*, contrairement par exemple aux constantes $\top$ et $\bot$ dont le sens est toujours fixé respectivement à 1 et 0.

Définition 4.3.4 (Signature, constante, symbole de fonction, variable propositionnelle, et relation) *Une* signature Σ *est un ensemble de déclarations de symboles de la forme s^{gn}. Soient n un entier strictement positif et Σ une signature, le symbole s est :*

1. *une* constante *de la signature si et seulement si $s^{f0} \in \Sigma$*
2. *un* symbole de fonction à *n* arguments *de la signature, si et seulement si $s^{fn} \in \Sigma$*
3. *une* variable propositionnelle *de la signature si et seulement si $s^{r0} \in \Sigma$*
4. *un* symbole de relation à *n* arguments *de la signature, si et seulement si $s^{rn} \in \Sigma$*

Au lieu de dire $0^{f0}, 1^{f0}, +^{f2}, *^{f2}, =^{r2}$ est une signature pour l'arithmétique, nous disons plus simplement qu'une signature possible pour l'arithmétique comporte 0, 1, $+$ (à deux arguments), $*$ et $=$. Sur cet exemple, nous devons préciser que le symbole *plus* est utilisé avec deux arguments (car nous pouvons aussi rencontrer le symbole *plus* avec un seul argument).

Exemple 4.3.5 *Une signature possible pour la théorie des ensembles est $\in, =$. Notons que toutes les autres opérations sur les ensembles peuvent être définies à partir de ces deux symboles.*

Nous définissons maintenant les termes associés à une signature. Nous considérons un ensemble dénombrable de variables disjoint de l'ensemble de symboles de la signature.

Définition 4.3.8 (Terme sur une signature) *Soit Σ une signature, un* terme *sur Σ est :*

— *soit une variable,*
— *soit une constante s où $s^{f0} \in \Sigma$,*

— *soit un terme de la forme* $s(t_1,\ldots,t_n)$*, où* $n \geq 1$*,* $s^{fn} \in \Sigma$ *et* $t_1,\ldots,t_n$ *sont des termes sur* Σ*.*

L'ensemble des termes sur la signature Σ est noté T_Σ.

Définition 4.3.9 (Formule atomique sur une signature) *Soit* Σ *une signature, une* formule atomique *sur* Σ *est :*
— *soit une des constantes* $\top, \bot$*,*
— *soit une variable propositionnelle* s *où* $s^{r0} \in \Sigma$*,*
— *soit de la forme* $s(t_1,\ldots,t_n)$ *où* $n \geq 1$*,* $s^{rn} \in \Sigma$ *et où* $t_1,\ldots,t_n$ *sont des termes sur* Σ*.*

Notons que les variables ne sont pas des formules atomiques.

Définition 4.3.10 (Formule sur une signature) *Une* formule *sur une signature* Σ *est une formule, dont les sous-formules atomiques sont des formules atomiques sur* Σ *(au sens de la définition 4.3.9).*

Nous dénotons l'ensemble des formules sur la signature Σ par F_Σ .

Exemple 4.3.11 $\forall x\ (p(x) \Rightarrow \exists y\ q(x,y))$ *est une formule sur la signature* $\Sigma = \{p^{r1}, q^{r2}, h^{f1}, c^{f0}\}$*. Mais c'est aussi une formule sur la signature* $\Sigma' = \{p^{r1}, q^{r2}\}$*, puisque les symboles h et c ne figurent pas dans la formule.*

Définition 4.3.12 (Signature associée à une formule) *La* signature associée *à une formule est la plus petite signature* Σ *telle que la formule est élément de* F_Σ*, c'est la plus petite signature permettant d'écrire la formule.*

Exemple 4.3.13 *La signature* Σ *associée à la formule* $\forall x\ (p(x) \Rightarrow \exists y\ q(x,y))$ *est* $\Sigma = \{p^{r1}, q^{r2}\}$*.*

Définition 4.3.14 (Signature associée à un ensemble de formules) *La* signature associée *à un ensemble de formules est l'union des signatures associées à chaque formule de l'ensemble.*

Exemple 4.3.15 *La signature* Σ *associée à l'ensemble constitué des deux formules* $\forall x(p(x) \Rightarrow \exists y\ q(x,y)), \forall u\ \forall v\ (u + s(v) = s(u) + v)$ *est* $\Sigma = \{p^{r1}, q^{r2}, +^{f2}, s^{f1}, =^{r2}\}$*.*

4.3.3 Interprétation

En logique propositionnelle, le sens des formules est uniquement fixé par les valeurs des variables, en logique du premier ordre le sens des formules dépend aussi du sens des fonctions et des relations. Le sens des fonctions et des relations est fixé par une interprétation.

Définition 4.3.16 (Interprétation) *Une* interprétation *I sur une signature* Σ *est définie par un domaine D non vide et une application qui à chaque déclaration de symbole* $s^{gn} \in \Sigma$ *associe sa valeur* s_I^{gn} *comme suit :*

1. s_I^{f0} *est un élément de D.*
2. s_I^{fn} *où* $n \geq 1$ *est une fonction de* D^n *dans D, autrement dit une fonction à n arguments.*
3. s_I^{r0} *vaut* 0 *ou* 1.
4. s_I^{rn} *où* $n \geq 1$ *est un sous-ensemble de* D^n*, autrement dit une relation à n arguments.*

Exemple 4.3.17 *Soit l'interprétation I de domaine* $D = \{1,2,3\}$ *où la relation binaire* ami *est vraie pour les couples* $(1,2)$*,* $(1,3)$ *et* $(2,3)$*, c'est-à-dire,* $ami_I^{r2} = \{(1,2),(1,3),(2,3)\}$*.* $ami(2,3)$ *est vraie dans l'interprétation I. En revanche,* $ami(2,1)$ *est fausse dans l'interprétation I.*

Remarque 4.3.18 *Dans* toute *interprétation I, la valeur du symbole* $=$ *est l'ensemble* $\{(d,d) \mid d \in D\}$*, autrement dit dans toute interprétation le sens de l'égalité est* l'identité *sur le domaine de l'interprétation.*

Exemple 4.3.19 *Considérons une signature suivante.*
— $Anne^{f0}$*,* $Bernard^{f0}$ *et* $Claude^{f0}$ *: les prénoms Anne, Bernard, et Claude dénotent des constantes,*
— a^{r2} *: la lettre a dénote une relation à deux arguments (nous lisons* $a(x,y)$ *comme x aime y) et*
— c^{f1} *: la lettre c dénote une fonction à un argument (nous lisons* $c(x)$ *comme le copain ou la copine de x).*
Une interprétation possible sur cette signature est l'interprétation I de domaine $D = \{0,1,2\}$ *où :*
— $Anne_I^{f0} = 0$*,* $Bernard_I^{f0} = 1$*, et* $Claude_I^{f0} = 2$*.*
— $a_I^{r2} = \{(0,1),(1,0),(2,0)\}$*.*

— $c_I^{f1}(0) = 1, c_I^{f1}(1) = 0, c_I^{f1}(2) = 2$. *Notons que la fonction* c_I^{f1} *a comme domaine D, ce qui oblige à définir artificiellement* $c_I^{f1}(2)$ *: Claude, dénoté par* 2*, n'a ni copain, ni copine.*

Définition 4.3.20 (Interprétation d'un ensemble de formules) L'interprétation d'un ensemble de formules *est une interprétation qui définit seulement le sens de la signature associée à l'ensemble des formules.*

Définition 4.3.21 (État) *Un* état *e d'une interprétation est une application de l'ensemble des variables dans le domaine de l'interprétation.*

Définition 4.3.22 (Assignation) *Une* assignation *est un couple* (I,e) *composé d'une interprétation I et d'un état e.*

Exemple 4.3.23 *Soient le domaine* $D = \{1,2,3\}$ *et l'interprétation I où la relation binaire* ami *est vraie uniquement pour les couples* $(1,2)$*,* $(1,3)$ *et* $(2,3)$*, c'est-à-dire,* $ami_I^{r2} = \{(1,2),(1,3),(2,3)\}$*. Soit e l'état qui associe* 2 *à x et* 1 *à y. L'assignation* (I,e) *rend la relation* $ami(x,y)$ *fausse.*

4.3.4 Sens des formules

Nous expliquons maintenant comment évaluer une formule à partir d'une assignation, c'est-à-dire, une interprétation et un état. Il faut noter que dans certains cas, l'état de l'assignation est inutile pour fixer le sens d'une formule.

Remarque 4.3.24 *La valeur d'une formule ne dépend que de ses variables libres et de ses symboles, aussi pour évaluer une formule sans variable libre, l'état des variables est inutile. Nous avons alors deux possibilités :*

— *Pour une formule* sans variables libres*, il suffit de donner une interprétation I des symboles de la formule. Dans ce cas, les assignations* (I,e) *et* (I,e') *donnerons la même valeur à la formule pour tous états e et* e'*. Ainsi pour tout état e, nous identifierons* (I,e) *et I. Selon le contexte, I sera considéré comme soit une interprétation soit une assignation dont l'état est quelconque.*

— *Pour une formule* avec des variables libres*, nous avons donc besoin d'une assignation.*

Soient Σ une signature, I une interprétation sur Σ de domaine D et e un état de cette interprétation. Nous souhaitons connaître la valeur (0 ou 1) de toute formule A sur Σ dans l'assignation (I,e). Cette valeur sera notée $[A]_{(I,e)}$. Pour cela, nous avons tout d'abord besoin de définir le sens de chaque terme t sur Σ dans l'assignation (I,e), noté $[\![t]\!]_{(I,e)}$. Ensuite nous devons fixer le sens de chaque formule atomique B sur Σ dans la même assignation, noté $[B]_{(I,e)}$.

4.3.4.1 Sens des termes sur une signature

Chacun sait intuitivement comment évaluer un terme : nous remplaçons les variables par leurs valeurs, les symboles de fonctions par les fonctions qui leur sont associées et nous appliquons les fonctions. Mais pour raisonner sur le sens des termes ou écrire un programme d'évaluation des termes, nous devons formaliser cette évaluation.

Définition 4.3.25 (Évaluation) *Nous donnons la définition inductive de l'évaluation d'un terme t :*

1. *si t est une variable, alors* $[\![t]\!]_{(I,e)} = e(t)$*,*
2. *si t est une constante alors* $[\![t]\!]_{(I,e)} = t_I^{f0}$*,*
3. *si* $t = s(t_1,\ldots,t_n)$ *où s est un symbole et* $t_1,\ldots,t_n$ *sont des termes, alors* $[\![t]\!]_{(I,e)} = s_I^{fn}([\![t_1]\!]_{(I,e)},\ldots,[\![t_n]\!]_{(I,e)})$*.*

Dans les exemples, nous remplaçons les variables par leurs valeurs, nous confondons les symboles et leur sens.

Exemple 4.3.26 *Soit la signature* a^{f0}, f^{f2}, g^{f2}*.*
Soit I l'interprétation de domaine $\mathbb{N}$ *où :*
— $a_I = 1$ *;*
— f_I^{f2} *est le produit ;*
— g_I^{f2} *est la somme.*
Soit e l'état tel que $x = 2, y = 3$*. Calculons* $[\![f(x,g(y,a))]\!]_{(I,e)}$*.*

4.3.4.2 Sens des formules atomiques sur une signature

Définition 4.3.27 (Sens des formules atomiques) *Le sens des formules atomiques est donné par les règles inductives suivantes :*

1. $[\top]_{(I,e)} = 1, [\bot]_{(I,e)} = 0$. *Dans les exemples, nous autorisons à remplacer* $\top$ *par sa valeur* 1 *et* $\bot$ *par sa valeur* 0.
2. *Soit s une variable propositionnelle,* $[s]_{(I,e)} = s_I^{r0}$.
3. *Soit* $A = s(t_1, \ldots, t_n)$ *où s est un symbole et* $t_1, \ldots, t_n$ *sont des termes. Si* $([\![t_1]\!]_{(I,e)}, \ldots, [\![t_n]\!]_{(I,e)}) \in s_I^{rn}$ *alors* $[A]_{(I,e)} = 1$ *sinon* $[A]_{(I,e)} = 0$.

Remarque 4.3.28 *Nous devons distinguer entre évaluer un terme* $[\![\ ]\!]$ *et évaluer une formule atomique* $[\ ]$, *car dans notre présentation, ces deux catégories syntaxiques ne sont pas disjointes, car* $s(t_1, \ldots, t_n)$ *peut être l'application d'une fonction* ($[\![\ ]\!]$) *ou d'une relation* ($[\ ]$). *Mais quand le contexte est sans ambiguïté, nous pouvons omettre cette distinction.*

Exemple 4.3.29 *En reprenant l'exemple* *4.3.19 page 79*, *nous obtenons :*

— $[a(Anne, Bernard)]_I =$

— $[a(Anne, Claude)]_I =$

Soit e l'état $x = 0, y = 2$. *Nous avons :*

— $[a(x, c(x))]_{(I,e)} =$

— $[a(y, c(y))]_{(I,e)} =$

Attention à distinguer (suivant le contexte), les éléments du domaine $0, 1$ *et les valeurs de vérité* $0, 1$. *Les exemples suivants mettent en évidence l'interprétation de l'égalité comme une identité. Dans l'interprétation I, nous avons :*

— $[(Anne = Bernard)]_I =$

— $[(c(Anne) = Anne)]_I =$

— $[(c(c(Anne)) = Anne)]_I =$

4.3.4.3 Sens des formules sur une signature

Nous pouvons maintenant ajouter le sens des quantificateurs et aussi celui des opérateurs qui reste identique à celui donné pour la logique propositionnelle (*cf.* sous-section 1.2.1 page 14).

Définition 4.3.30 (Sens des formules) *Le sens des formules est donné par :*

1. *Les connecteurs propositionnels ont le même sens qu'en logique propositionnelle. Soient B et C des formules, rappelons uniquement le sens de l'implication : si* $[B]_{(I,e)} = 0$ *alors* $[(B \Rightarrow C)]_{(I,e)} = 1$ *sinon* $[(B \Rightarrow C)]_{(I,e)} = [C]_{(I,e)}$.

2. *Soient x une variable et B une formule.* $[\forall xB]_{(I,e)} = 1$ *si et seulement si* $[B]_{(I,f)} = 1$ pour tout *état f identique à e, sauf pour x. Soit* $d \in D$. *Notons* $e[x = d]$ *l'état identique à l'état e, sauf pour la variable x, auquel l'état* $e[x = d]$ *associe la valeur d. La définition ci-dessus peut être mise sous la forme suivante :*

$$[\forall xB]_{(I,e)} = min_{d\in D}[B]_{(I,e[x=d])} = \prod_{d\in D}[B]_{(I,e[x=d])},$$

où le produit est le produit booléen.
Cette définition permet de calculer la valeur de $\forall xB$ *dans le cas où D est un domaine fini. Mais quand D n'est pas un domaine fini, c'est seulement une traduction du quantificateur universel d'un formalisme dans un autre.*

3. $[\exists xB]_{(I,e)} = 1$ *si et seulement s'*il y a *un état f identique à e, sauf pour x, tel que* $[B]_{(I,f)} = 1$. *La définition ci-dessus peut être mise sous la forme suivante :*

$$[\exists xB]_{(I,e)} = max_{d\in D}[B]_{(I,e[x=d])} = \sum_{d\in D}[B]_{(I,e[x=d])},$$

où la somme est la somme booléenne.

Exemple 4.3.31 *Soit l'interprétation I de domaine* $D = \{1,2,3\}$ *où la relation binaire* ami *est vraie pour les couples* $(1,2)$, $(1,3)$ *et* $(2,3)$, *c'est-à-dire,* $ami_I^r{}^2 = \{(1,2),(1,3),(2,3)\}$. *La formule* $ami(1,2) \wedge ami(2,3) \Rightarrow ami(1,3)$ *est vraie dans l'interprétation I,* i.e., $[ami(1,2) \wedge ami(2,3) \Rightarrow ami(1,3)]_I = 1$.

Exemple 4.3.32 *Utilisons l'interprétation I donnée dans l'exemple* *4.3.19 page 79*. *D'après le sens du quantificateur existentiel, nous devons évaluer* $a(x,x)$ *pour* $x = 0$, $x = 1$, *et* $x = 2$. *Nous simplifions ce calcul et les calculs suivants en remplaçant immédiatement les variables par leurs valeurs : cela nous évite d'utiliser les états.*

— $[\exists x\ a(x,x)]_I =$

— $[\forall x \exists y\ a(x,y)]_I = \min\{\max\{[a(0,0)]_I, [a(0,1)]_I, [a(0,2)]_I\}, \max\{[a(1,0)]_I, [a(1,1)]_I, [a(1,2)]_I\}, \max\{[a(2,0)]_I, [a(2,1)]_I, [a(2,2)]_I\}\} = \min\{\max\{0,1,0\}, \max\{1,0,0\}, \max\{1,0,0\}\} = \min\{1,1,1\} = 1$.
D'après la définition, nous avons : $[\forall x \exists y\ a(x,y)]_I = ([a(0,0)]_I + [a(0,1)]_I + [a(0,2)]_I).\ ([a(1,0)]_I + [a(1,1)]_I + [a(1,2)]_I).\ ([a(2,0)]_I + [a(2,1)]_I + [a(2,2)]_I) = (0+1+0).(1+0+0).(1+0+0) = 1.1.1 = 1$.

— $[\exists y \forall x\ a(x,y)]_I =$

Remarque 4.3.33 *Dans l'interprétation ci-dessus, les formules* $\forall x \exists y\ a(x,y)$ *et* $\exists y \forall x\ a(x,y)$ *n'ont pas la même valeur. Donc en intervertissant un quantificateur existentiel et un quantificateur universel, nous ne préservons pas le sens des formules.*

4.3.5 Modèle, validité, conséquence, équivalence

Ces notions sont définies comme en logique propositionnelle. Mais alors qu'en logique propositionnelle, une assignation est une application des variables propositionnelles dans $0,1$, en logique du premier ordre une assignation est un couple constitué d'une interprétation des symboles d'une part et de l'état des variables d'autre part.

4.3.6 Instanciation

Définition 4.3.34 (Instanciation) *Soit x une variable, t un terme et A une formule.*

1. $A<x := t>$ *est la formule obtenue en remplaçant dans la formule A toute occurrence libre de x par le terme t.*

2. *Le terme t est libre pour x dans A si les variables de t ne sont pas liées dans les occurrences libres de x dans A.*

Exemple 4.3.35 *Le terme z est libre pour x dans la formule $\exists y p(x,y)$. Par contre le terme y, comme tout terme comportant la variable y, n'est pas libre pour x dans cette formule. Soit A la formule $(\forall x P(x) \vee Q(\mathbf{x}))$, la formule $A{<}x := b{>}$ vaut*

Théorème 4.3.36 *Soient A une formule et t un terme libre pour la variable x dans A. Soient I une interprétation et e un état de l'interprétation. Nous avons $[A{<}x := t{>}]_{(I,e)} = [A]_{(I,e[x=d])}$, où $d = [\![t]\!]_{(I,e)}$.*

Autrement dit, la valeur de $A{<}x := t{>}$ dans une assignation est la même que celle de A dans une assignation identique, sauf qu'elle donne à x la valeur du terme t. Ce théorème, dont le résultat est évident, peut être prouvé par une récurrence (que nous ne ferons pas) sur la taille des formules. Nous montrons seulement que la condition sur t est indispensable en observant un exemple où cette condition n'est pas respectée.

Exemple 4.3.37 *Soient I l'interprétation de domaine $\{0,1\}$ avec $p_I = \{(0,1)\}$ et e, un état où $y = 0$. Soient A la formule $\exists y p(x,y)$ et t le terme y.* Ce terme n'est pas libre pour x dans A. *Nous avons :*

— $A{<}x := y{>}=$

et $[A{<}x := y{>}]_{(I,e)} =$

— *Soit $d = [\![t]\!]_{(I,e)} = [\![y]\!]_{(I,e)} = 0$. Dans l'assignation $(I, e[x = d])$, nous avons $x = 0$. Donc, nous obtenons :*
$[A]_{(I,e[x=d])} =$

Ainsi, $[A{<}x := t{>}]_{(I,e)} \neq [A]_{(I,e[x=d])}$, pour $d = [\![t]\!]_{(I,e)}$.

Corollaire 4.3.38 *Soient A une formule et t un terme libre pour x dans A. Les formules $\forall x A \Rightarrow A{<}x := t{>}$ et $A{<}x := t{>} \Rightarrow \exists x A$ sont valides.*

Ce corollaire est une conséquence immédiate du théorème précédent.

4.3.7 Interprétation finie

Nous montrons comment rechercher des modèles finis de formules fermées (c'est-à-dire sans variables libres). Un modèle fini d'une formule fermée est une interprétation de la formule de domaine fini, qui rend vraie la formule. Il est clair que le nom des éléments du domaine est sans importance, aussi quand nous cherchons un modèle avec un domaine de n éléments, le domaine que nous utiliserons, sera celui des entiers (sous-entendu naturels) inférieurs à n. Pour savoir si une formule fermée a un modèle de domaine $\{0,\ldots,n-1\}$, il suffit d'énumérer toutes les interprétations possibles de la signature associée à la formule et d'évaluer la formule pour ces interprétations. Mais cette méthode est inutilisable en pratique, car le nombre de ces interprétations est énorme. Soit une signature avec une constante, un symbole de fonction à un argument et un symbole de relation à deux arguments (plus éventuellement l'égalité de sens fixé), sur un domaine à 5 éléments, cette signature à $5 \times 5^5 \times 2^{25} = 524288000000$ interprétations. Aussi nous montrons d'abord, comment dans le cas *où une formule n'a aucun symbole de fonction et aucune constante, sauf des représentations d'entiers inférieurs à n*, nous pouvons rechercher ses modèles à n éléments par réduction au cas propositionnel. En présence de symboles de fonctions et de constantes, nous pouvons en énumérer leurs valeurs, puis appliquer les méthodes ci-dessous. Pour trouver des énumérations intelligentes, nous conseillons de lire le manuel de Prover9 [22].

4.3.7.1 Les entiers et leurs représentations

Nous distinguons dans la suite un entier n et sa représentation $\underline{n}$. Par exemple l'entier 3 peut être représenté en base 10 par 3, en base 2 par 11, en chiffres romains par III, en chiffre grec par γ. Par habitude, nous choisissons la représentation en base 10. Dans la suite, de façon implicite, toutes les interprétations considérées donnent à la représentation d'un entier, la valeur de l'entier représenté.

4.3.7.2 Expansion d'une formule

Dans certains cas il est suffisant de regarder les expansions de taille finie afin de trouver un modèle ou un contre-modèle pour une formule donnée.

Définition 4.3.39 (n-expansion) *Soient A une formule et n un entier. La n-*expansion de *A est la formule qui consiste à remplacer toute sous-formule de A de la forme $\forall xB$ par la conjonction $(\prod_{i<n} B{<}x := \underline{i}{>})$ et toute sous-formule de A de la forme $\exists xB$ par la disjonction $(\sum_{i<n} B{<}x := \underline{i}{>})$ où $\underline{i}$ est la représentation décimale de l'entier i.*

Exemple 4.3.40 *La 2-expansion de la formule $\exists xP(x) \Rightarrow \forall xP(x)$ est la formule*

Théorème 4.3.41 *Soient n un entier et A une formule ne comportant que des représentations d'entiers de valeur inférieure à n. Soit A' la n-expansion de A. Toute interprétation de domaine $\{0,\ldots,n-1\}$ attribue la même valeur à A et à A'.*

La condition sur A est nécessaire car si A comporte une représentation d'un entier au moins égal à n, la valeur de cette représentation ne sera pas dans le domaine de l'interprétation. La preuve du théorème est une récurrence sur la taille des formules, que nous ne ferons pas. Nous nous contentons de montrer que l'élimination d'un quantificateur universel de la formule $\forall xB$ conserve la valeur de cette formule.

Soient (I,e) une interprétation et un état de domaine $\{0,\ldots,n-1\}$ donnant à la représentation d'un entier, la valeur de l'entier représenté. Par définition du sens du quantificateur universel : $[\forall xB]_{(I,e)} = \prod_{i<n}[B]_{(I,e[x=i])}$. D'après le théorème 4.3.36 page précédente et le fait que la valeur de la représentation de l'entier i est i, nous avons : $[B]_{(I,e[x=i])} = [B{<}x := \underline{i}{>}]_{(I,e)}$. Par suite : $[\forall xB]_{(I,e)} = \prod_{i<n}[B{<}x := \underline{i}{>}]_{(I,e)} = [\prod_{i<n} B{<}x := \underline{i}{>}]_{(I,e)}$.

4.3.7.3 Interprétation et assignation propositionnelle

Soient n un entier et A une formule fermée, sans quantificateur, sans égalité, sans symbole de fonction, sans constante sauf des représentations d'entiers inférieurs à n. Soit P l'ensemble des formules atomiques de A (sauf $\top$ et $\bot$ dont le sens est fixé).

De l'assignation à l'interprétation.

Théorème 4.3.42 *Soit v une assignation propositionnelle de P dans $\{0,1\}$ alors il existe une interprétation I de A telle que $[A]_I = [A]_v$.*

Preuve : Soit v une assignation propositionnelle de P dans $\{0,1\}$. Soit I l'interprétation suivante de A :

1. Domaine : $\{0,\ldots,n-1\}$
2. $s_I^{r0} = v(s)$ si et seulement si $s \in P$
3. Soit $p \geq 1$ et s^{rp} un symbole de relation de A . Alors $s_I^{rp} = \{(k_1,\ldots,k_p) | s(\underline{k_1},\ldots,\underline{k_p}) \in P \text{ et } v(s(\underline{k_1},\ldots,\underline{k_p})) = 1\}$

Par définition de I, l'assignation v et l'interprétation I donnent la même valeur aux sous-formules atomiques de A, donc (par récurrence sur les formules) la même valeur à la formule A. □

Exemple 4.3.43 *L'assignation v, définie par $p(0) = 1$ et $p(1) = 0$, donne la valeur 0 à la formule suivante :*

$$(p(0) + p(1)) \Rightarrow (p(0).p(1)).$$

Donc I définie par $p_I = \{0\}$ donne aussi la valeur 0 à cette même formule. Cet exemple montre que v et I sont deux façons analogues de présenter une interprétation, la deuxième étant souvent plus concise.

De l'interprétation à l'assignation.

Théorème 4.3.44 *Soit I une interprétation de A alors il existe une assignation v de P telle que*

$$[A]_I = [A]_v.$$

Preuve : Soient I une interprétation de A et v l'assignation propositionnelle suivante de P dans $\{0,1\}$: pour tout $B \in P$, $v(B) = [B]_I$. Par définition de v, l'assignation v et l'interprétation I donnent la même valeur aux sous-formules atomiques de A, donc la même valeur à la formule A. □

4.3.7.4 Recherche d'un modèle fini d'une formule fermée

Formule fermée sans symbole de fonction : Soit A une formule fermée sans symbole de fonction ni constantes, sauf des représentations d'entiers de valeur inférieure à n. Pour trouver une interprétation I modèle de A de domaine $\{0,\ldots,n-1\}$ donnant à la représentation d'un entier, la valeur de l'entier représenté, nous procédons ainsi :

1. Nous remplaçons A par sa n-expansion B.
2. Dans la formule B, nous remplaçons les égalités par leur valeur, c'est-à-dire que $\underline{i} = \underline{j}$ est remplacée par 0 si $i \neq j$ et par 1 si $i = j$. De plus, nous recommandons d'éliminer ces valeurs de vérité par les identités $x+0=x$, $x+1=1$, $x*0=0$, $x*1=x$. Soit C la formule obtenue après ces remplacements et simplifications.
3. Nous cherchons une assignation propositionnelle v des formules atomiques de C, qui soit modèle de C : si une telle assignation n'existe pas, A n'a pas de modèle, sinon l'interprétation I déduite de v comme il est indiqué dans le théorème 4.3.42 page précédente est modèle de A.

Prouvons la correction de cette méthode :

1. Supposons qu'il n'y ait pas d'assignation propositionnelle modèle de C, mais que A ait un modèle I. D'après le théorème 4.3.41 page ci-contre, I est modèle de B, donc de C, et d'après le théorème 4.3.44 page précédente, il y a une assignation propositionnelle modèle de C. De cette contradiction, nous déduisons que A n'a pas de modèle à n éléments.
2. Supposons que l'assignation propositionnelle v soit modèle de C. Donc, l'interprétation I construite comme il est indiqué dans le théorème 4.3.42 page ci-contre est modèle de C, donc elle est modèle de B, donc aussi d'après le théorème 4.3.41 page précédente, elle est modèle de A.

Exemple 4.3.45 *Soit A la formule $\exists x P(x) \wedge \exists x \neg P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y)$. Il est clair que cette formule n'a pas de modèle à un élément car cet élément devrait vérifier à la fois la propriété P et sa négation. La 2-expansion de A est :*

Formule fermée avec symbole de fonction : Soit A une formule fermée pouvant comporter des représentations d'entiers de valeur inférieure à n. Comme dans le cas précédent, nous remplaçons A par son expansion, nous supprimons les égalités. Puis, nous énumérons les choix des valeurs des symboles comme dans *l'algorithme* `DPLL`, en propageant le plus possible chacun des choix effectués.

Exemple 4.3.46 *Soit A la formule $\exists y\, P(y) \Rightarrow P(a)$, dont nous cherchons un contre-modèle à 2 éléments (cela revient au même que de trouver un modèle de la négation de A).*

Exemple 4.3.47 *Nous cherchons un modèle à 2 éléments des formules* $P(a)$, $\forall x(P(x) \Rightarrow P(f(x)))$, $\neg P(f(b))$.

4.3.8 Substitution et remplacement

La propriété de substitution et la propriété de remplacement déjà énoncées en logique propositionnelle (voir 1.3.1 page 19 et 1.3.10 page 21) s'étendent à la logique du premier ordre[3]. Plus précisément l'application d'une substitution à une formule *propositionnellement* valide donne une formule valide. Par exemple soit σ la substitution propositionnelle telle que $\sigma(p) = \forall x\ q(x)$. Puisque la formule $p \vee \neg p$ est valide, il en est de même de la formule $\sigma(p \vee \neg p) = \forall x\ q(x) \vee \neg\forall x\ q(x)$. Le principe de remplacement s'étend aussi avec le même énoncé de la logique propositionnelle à la logique du premier ordre car il est déduit des propriétés élémentaires suivantes. Pour toutes formules A et B et toute variable x :

— $(A \Leftrightarrow B) \models (\forall xA \Leftrightarrow \forall xB)$.
— $(A \Leftrightarrow B) \models (\exists xA \Leftrightarrow \exists xB)$.

4.4 Équivalences remarquables

Dans cette section, nous donnons les règles permettant de simplifier les formules de la logique du premier ordre. Remarquons que les règles de simplifications de la logique propositionnelle sont bien entendu encore appliquables au premier ordre.

4.4.1 Relation entre $\forall$ et $\exists$

Lemme 4.4.1 *Soient A une formule et x une variable.*

1. $\neg\forall xA \equiv \exists x\neg A$.

3. Notons que dans le livre de Stephen Cole Kleene [19], cette notion de substitution sur les variables propositionnelles est étendue à une substitution plus générale applicable à tous les symboles de relation.

2. $\forall x A \equiv \neg\exists x \neg A$.
3. $\neg\exists x A \equiv \forall x \neg A$.
4. $\exists x A \equiv \neg\forall x \neg A$.

Prouvons les deux premières identités, les deux autres sont données en exercice 78 page 92 :

1. Soient I une interprétation de domaine D et e un état.

$$\begin{array}{rcll} [\neg\forall x A]_{(I,e)} & \equiv & 1-[\forall x A]_{(I,e)} & \\ & \equiv & 1-\prod_{d\in D}[A]_{(I,e[x=d])} & \text{d'après le sens de } \forall \\ & \equiv & \sum_{d\in D}\left(1-[A]_{(I,e[x=d])}\right) & \text{par les lois de De Morgan généralisées} \\ & \equiv & \sum_{d\in D}[\neg A]_{(I,e[x=d])} & \text{par définition du sens de } \neg \\ & \equiv & [\exists x \neg A]_{(I,e)} & \text{par définition du sens de } \exists \end{array}$$

En examinant de façon critique cette preuve, nous voyons qu'elle utilise la loi de De Morgan généralisée, qui est l'expression dans un autre formalisme d'une loi analogue. Il faut donc formaliser l'usage des quantificateurs pour échapper à ces preuves qui ne sont en fait que des changements de notations. Nous le ferons en ajoutant à la déduction naturelle des règles qui gouvernent les quantificateurs.

2. Les autres équivalences sont prouvables à partir de la première par le principe de remplacement. Nous ne prouvons que la deuxième identité :

$$\begin{array}{rcll} \forall x A & \equiv & \neg\neg\forall x A & \text{identité de la double négation} \\ & \equiv & \neg\exists x \neg A & \text{par l'identité 1 page précédente} \end{array}$$

4.4.2 Déplacement des quantificateurs

Nous donnons, sans preuve, des équivalences qui s'appliquent à toutes les formules (avec ou sans variable libre) et qui permettent de retrouver les lois usuelles de déplacement des quantificateurs. Nous supposons ci-dessous que x, y sont des variables et que A, B sont des formules.

1. $\forall x \forall y A \equiv \forall y \forall x A$.
2. $\exists x \exists y A \equiv \exists y \exists x A$.
3. $\forall x (A \wedge B) \equiv (\forall x A \wedge \forall x B)$.
4. $\exists x (A \vee B) \equiv (\exists x A \vee \exists x B)$.
5. Soient Q un des quantificateurs $\forall, \exists$, et $\circ$ une des opérations $\wedge, \vee, \Rightarrow$. Supposons que x ne soit pas une variable libre de A.
   (a) $QxA \equiv A$,
   (b) $Qx(A \circ B) \equiv (A \circ QxB)$.

Exemple 4.4.2 *Nous éliminons de ces deux formules les quantificateurs inutiles :*

— $\forall x \exists x P(x) \equiv$

— $\forall x (\exists x P(x) \vee Q(x)) \equiv$

4.4.3 Changement de variables liées

Théorème 4.4.3 *Soit Q un des quantificateurs $\forall, \exists$. Supposons que y soit une variable qui ne figure pas dans QxA alors : $QxA \equiv QyA<x := y>$.*

Exemple 4.4.4 *La formule $\forall x\ p(x,z)$ est équivalente à la formule $\forall y\ p(y,z)$ (d'après le théorème ci-dessus) mais elle n'est pas équivalente à la formule $\forall z\ p(z,z)$, où le changement de variable ne respecte pas les conditions du théorème.*

Définition 4.4.5 (Formules égales à un changement près de variables liées) *Deux formules sont* égales à un changement près de variables liées *si nous pouvons obtenir l'une à partir de l'autre par des remplacements de sous-formules de la forme QxA par QyA<x := y> où Q est un quantificateur et y est une variable qui ne figure pas dans QxA. Deux formules égales à un changement près de variables liées, sont dites aussi copies l'une de l'autre ou encore* α-équivalentes.

Théorème 4.4.6 *Si deux formules sont égales à un changement près de variables liées alors elles sont équivalentes.*

Ce résultat est clairement une conséquence du théorème 4.4.3 page précédente. Nous n'en ferons pas la preuve, qui est longue et fastidieuse. Nous nous contentons de suggérer cette preuve à l'aide d'un exemple.

Exemple 4.4.7 *Nous montrons que les formules* $\forall x \exists y P(x,y)$ *et* $\forall y \exists x P(y,x)$ *sont égales par changement des variables liées, au sens de la définition 4.4.3 page précédente donc sont équivalentes, en effet :*

Notre définition de l'égalité entre deux formules à un changement près de variables liées n'est pas pratique, car la définition ne donne pas un test. Or il est simple de voir si deux formules sont égales à un changement près de variables liées : nous traçons des traits entre chaque quantificateur et les variables qu'il lie et nous effaçons les noms des variables liées. Si après cette transformation, les deux formules deviennent identiques, c'est qu'elles sont égales à un changement près des variables liées.

Exemple 4.4.8 *Les deux formules* $\forall x \exists y P(y,x)$ *et* $\forall y \exists x P(x,y)$ *sont transformées en* $\forall \underline{|} \exists \underline{|} \underline{P(|,|)}$.

4.5 Exercices

Exercice 64 (Structure et variables libres)
Pour chaque formule ci-dessous, indiquer sa structure et ses variables libres.

1. $\forall x(P(x) \Rightarrow \exists y Q(x,y))$.
2. $\forall a \forall b(b \neq 0 \Rightarrow \exists q \exists r(a = b * q + r \wedge r < b))$[4].
3. $Pair(x) \Leftrightarrow \exists y(x = 2 * y)$.
4. x *Divise* $y \Leftrightarrow \exists z(y = z * x)$.
5. $Premier(x) \Leftrightarrow \forall y(y$ *Divise* $x \Rightarrow y = 1 \vee y = x)$.

□

Exercice 65 (Formalisation, symbole de fonction et de relation) *Nous considérons* $\Sigma = \{f^{r2}, o^{r2}, c^{r2}, j^{r2}, r^{f0}, p^{f1}\}$ *la signature ayant la sémantique donnée ci-dessous.*

— $f(x,y)$: *x est frère de y.*
— $o(x,y)$: *x est l'oncle de y.*
— $c(x,y)$: *x est le cousin de y.*
— $j(x,y)$: *x est plus jeune que y.*
— *r est le diminutif de Robert.*
— $p(x)$ *est le père de x.*

Exprimer en logique du premier ordre et en utilisant la signature Σ *les phrases suivantes :*

1. *Tout frère du père de Robert est un oncle de Robert.*
2. *Si les pères de deux enfants sont des frères alors ces deux enfants sont des cousins.*
3. *Robert a un cousin plus jeune qu'un des frères de Robert.*

Exprimer en français les propositions logiques suivantes :

1. $\nexists x\ j(p(x),x)$
2. $\forall x \forall y(p(p(x)) = p(p(y)) \Rightarrow c(x,y))$
3. $\forall x \exists y(f(x,y) \wedge j(x,y))$
4. $\exists x \exists y(f(x,y) \wedge \neg(p(x) = p(y)))$

□

Exercice 66 (Formalisation) *Considérons la signature* $\Sigma = \{a^{f0}, f^{f0}, J^{r2}, G^{r2}\}$, *où les symboles ont le sens donné ci-dessous.*

— a : *l'équipe d'Allemagne.*
— f : *l'équipe de France.*
— $J(x,y)$: *x a joué un match contre y.*
— $G(x,y)$: *x a gagné contre y.*

Exprimer en logique du premier ordre en utilisant la signature Σ *les assertions suivantes :*

1. *L'équipe de France a gagné un match et en a perdu un.*
2. *L'équipe de France et l'équipe d'Allemagne ont fait match nul.*
3. *Une équipe a gagné tous ses matchs.*
4. *Aucune équipe n'a perdu tous ses matchs.*
5. *Considérons l'assertion suivante : « Tous ceux qui ont joué contre une équipe qui a gagné tous ses matchs, ont gagné au moins un match ». Parmi les formules suivantes, lesquelles expriment la phrase ci-dessus, et lesquelles sont équivalentes ?*
   (a) $\forall x \exists y\,(J(x,y) \wedge \forall z(J(y,z) \Rightarrow G(y,z)) \Rightarrow \exists v G(x,v))$.
   (b) $\forall x(\exists y\,(J(x,y) \wedge \forall z(J(y,z) \Rightarrow G(y,z))) \Rightarrow \exists v G(x,v))$.
   (c) $\exists x(\forall y\,(J(x,y) \Rightarrow G(x,y)) \Rightarrow \forall z(J(x,z) \Rightarrow \exists v G(x,v)))$.
   (d) $\forall x \forall y\,(J(x,y) \wedge \forall z(J(y,z) \Rightarrow G(y,z)) \Rightarrow \exists v G(x,v))$.
   (e) $\forall x(\forall y\,(J(x,y) \wedge \forall z(J(y,z) \Rightarrow G(y,z))) \Rightarrow \exists v G(x,v))$.

4. Afin de respecter la notation usuelle pour la division euclidienne nous prenons exceptionnellement a, b, q et r comme nom de variable.

□

Exercice 67 (Formaliser,)** *Soient les constantes s pour Serge, t pour Tobby et les symboles de relation $A(x,y)$, x aime y, $C(x)$, x est un chien, $D(x)$, x est un animal domestique, $E(x)$, x est un enfant, $O(x)$, x est un oiseau et $P(x,y)$, x a peur de y, donner la signature Σ associée à ces symboles. Formaliser en logique du premier ordre en utilisant Σ les énoncés suivants :*

1. *Les chiens et les oiseaux sont des animaux domestiques.*
2. *Tobby est un chien qui aime les enfants.*
3. *Les oiseaux n'aiment pas les chiens.*
4. *Serge aime tous les animaux domestiques sauf les chiens.*
5. *Tous les enfants n'ont pas peur des chiens.*
6. *Certains chiens aiment les enfants.*
7. *Certains chiens aiment les enfants et réciproquement.*
8. *Les enfants aiment certains chiens.*

□

Exercice 68 (Évaluer des prédicats unaires) *Soit I l'interprétation de domaine $D = \{0,1\}$ telle que $P_I = \{0\}, Q_I = \{1\}$.*

1. *Évaluer dans cette interprétation les formules $\forall x P(x)$ et $\forall x(P(x) \vee Q(x))$.*
2. *Les formules $\forall x P(x) \vee \forall x Q(x)$ et $\forall x(P(x) \vee Q(x))$ sont-elles équivalentes ?*
3. *Évaluer dans cette interprétation les formules $\exists x P(x)$ et $\exists x(P(x) \wedge Q(x))$.*
4. *Les formules $\exists x P(x) \wedge \exists x Q(x)$ et $\exists x(P(x) \wedge Q(x))$ sont-elles équivalentes ?*
5. *Évaluer dans cette interprétation les formules $\forall x(P(x) \Rightarrow Q(x))$ et $\forall x P(x) \Rightarrow \forall x Q(x)$.*
6. *Les deux formules $\forall x(P(x) \Rightarrow Q(x))$ et $\forall x P(x) \Rightarrow \forall x Q(x)$ sont-elles équivalentes ?*

□

Exercice 69 (Interprétation,*) *Nous donnons les formules suivantes :*

1. $\forall x \exists y(y = x+1)$.
2. $\exists y \forall x(y = x+1)$.
3. $\forall x \exists y(y = x+1) \Rightarrow \exists y \forall x(y = x+1)$.
4. $\forall x \exists y(x = y+1)$.
5. $\exists x \forall y(y = x+y)$.
6. $\exists x(x \neq 0 \wedge x+x = x)$.

Nous donnons les interprétations suivantes :

1. *I1 est l'algèbre de Boole sur $\{0,1\}$.*
2. *I2 est l'arithmétique usuelle sur les entiers naturels.*
3. *I3 est l'arithmétique usuelle sur les entiers relatifs.*
4. *I4 est l'algèbre de Boole de domaine $\mathcal{P}(X)$ où les constantes 0 et 1 dénotent les ensembles $\emptyset$ et X et où l'addition est l'union d'ensembles.*

Indiquer si ces interprétations sont des modèles ou des contre-modèles des formules ci-dessus. □

Exercice 70 (Prédicat unaire et égalité) *Soit I l'interprétation de domaine $D = \{0,1,2\}$ telle que : $P_I = \{0,1\}, Q_I = \{1,2\}, R_I = \{\}$. Évaluer dans cette interprétation les formules suivantes :*

1. $\exists x R(x)$.
2. $\forall x(P(x) \vee Q(x))$.
3. $\forall x(P(x) \Rightarrow Q(x))$.
4. $\forall x(R(x) \Rightarrow Q(x))$.
5. $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$[5].

5. Cette formule signifie qu'il y a un et un seul élément vérifiant P.

6. $\exists x(P(x) \wedge Q(x) \wedge \forall y(P(y) \wedge Q(y) \Rightarrow x = y))$.

□

Exercice 71 (Évaluation, égalité) *Nous utilisons le symbole de fonction unaire f et la constante a. Nous abrégeons $\neg(x = y)$ en $x \neq y$. Soient I_1, I_2 les interprétations suivantes de domaine $\{0,1,2\}$: $a_{I_1} = a_{I_2} = 0$.*

x	$f_{I_1}(x)$	$f_{I_2}(x)$
0	0	1
1	0	2
2	2	0

Dans l'interprétation I_1 puis dans I_2, évaluer les formules suivantes :

1. $f(a) = a$.
2. $f(f(a)) = a$.
3. $f(f(f(a)) = a$.
4. $\exists x(f(x) = x)$, *f a un point-fixe.*
5. $\forall x(f(f(f(x))) = x)$.
6. $\forall y \exists x(f(x) = y)$, *$f$ est surjective.*
7. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$, *$f$ est injective.*
8. $\neg(\exists x \exists y(f(x) = f(y) \wedge x \neq y))$.

□

Exercice 72 (Formalisation et évaluation) *Nous adoptons les notations suivantes :*

— *Anatoli, Boris, Catarina et Denka sont des constantes du domaine,*
— *$P(x)$ signifie que x a réussi son examen,*
— *$Q(x,y)$ signifie que x a téléphoné à y.*

Donner la signature associée à ces notations et traduire en formules les énoncés suivants :

1. *Quelqu'un a raté l'examen et n'a été appelé par personne.*
2. *Tous ceux qui ont réussi à l'examen ont été appelés.*
3. *Personne n'a appelé tous ceux qui ont réussi à l'examen.*
4. *Tous ceux qui ont appelé quelqu'un, ont appelé quelqu'un qui a réussi l'examen.*

Soit l'interprétation ayant pour domaine $D = \{0,1,2,3\}$. Anatoli, Boris, Catarina et Denka valent respectivement 0, 1, 2 et 3 dans l'interprétation. Anatoli et Boris sont des garçons et Catarina et Denka sont des filles. Dans cette interprétation seuls Boris et Catarina ont réussi l'examen, les garçons ont appelé les filles, Denka a appelé Boris, Catarina a appelé Denka et ce sont les seuls appels.

Nous demandons de définir formellement l'interprétation donnée ci-dessus, puis de donner la valeur des énoncés précédents dans cette interprétation.

Indication *: pour faciliter le calcul de la valeur, nous suggérons de dessiner l'interprétation en entourant les prénoms des personnes qui ont réussi leurs examens, et en mettant une flèche de x vers y si x a téléphoné à y.* □

Exercice 73 (Expansion et contre-modèle) *Trouver, par la méthode des expansions, des contre-modèles des formules suivantes :*

1. $\exists x P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$.
2. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow \exists x Q(x)$.
3. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\exists x P(x) \Rightarrow \forall x Q(x))$.
4. $(\exists x F(x) \Rightarrow \exists x G(x)) \Rightarrow \forall x(F(x) \Rightarrow G(x))$.
5. $\forall x \exists y R(x,y) \Rightarrow \exists x R(x,x)$.
6. $\forall x \forall y(R(x,y) \Rightarrow R(y,x)) \Rightarrow \forall x R(x,x)$.

Indication *: il suffit de construire des 1 ou 2 expansions.* □

Exercice 74 (Raisonnement incorrect) *Considérons les hypothèses suivantes :*

1. $\exists x P(x)$.
2. $\exists x Q(x)$.
3. $\forall x(P(x) \wedge Q(x) \Rightarrow R(x))$.

Montrer, en utilisant la méthode des expansions, qu'il est incorrect de déduire à partir de ces trois hypothèses la conclusion suivante : $\exists x R(x)$. □

Exercice 75 (Contre-modèles avec relation) *Construire des contre-modèles des formules suivantes, où F est une relation :*

1. $\forall x \exists y(x = y) \Rightarrow \exists y \forall x(y = x)$.
2. $F(a) \wedge (a \neq b) \Rightarrow \neg F(b)$.
3. $\exists x \exists y(F(x) \wedge F(y) \wedge x \neq y) \Rightarrow \forall x F(x)$.
4. $\forall x \forall y(F(x,y) \Rightarrow x = y) \Rightarrow \exists x F(x,x)$.

□

Exercice 76 (Contre-modèles avec fonction) *Construire, en utilisant la méthode des expansions, des contre-modèles des formules suivantes, où f est une fonction et P une relation :*

1. $\forall y \exists x(f(x) = y)$.
2. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$.
3. $\exists x \forall y(f(y) = x)$.
4. $\forall x(P(x) \Rightarrow P(f(x))$.

□

Exercice 77 (Équivalences) *Prouver que :*

1. $\neg \forall x \exists y P(x,y) \equiv \exists y \forall x \neg P(y,x)$.
2. $\exists x(P(x) \Rightarrow Q(x)) \equiv \forall x P(x) \Rightarrow \exists x Q(x)$.
3. *La phrase « aucun malade n'aime les charlatans » a été traduite en logique du premier ordre par deux étudiants par les 2 formules suivantes :*
   — $\forall x \forall y((M(x) \wedge A(x,y)) \Rightarrow \neg C(y))$.
   — $\neg(\exists x(M(x) \wedge (\exists y(A(x,y) \wedge C(y)))))$.
   Montrer que ces étudiants disent la même chose, c'est-à-dire les deux formules associées aux traductions sont équivalentes.

□

Exercice 78 (☞, Preuve,*) *Prouver les deux équivalences suivantes du lemme 4.4.1 page 86 :*
— $\neg \exists x A \equiv \forall x \neg A$.
— $\exists x A \equiv \neg \forall x \neg A$.
Conseil *: utiliser les propriétés 1 page 86 et 2 page 87 du lemme 4.4.1 page 86.* □

Exercice 79 (Preuve) *Nous savons que $(\forall x(A \wedge B)) \equiv (A \wedge (\forall x B))$ à la condition que x ne soit pas une variable libre de A comme il est dit au paragraphe 4.4.2 page 87. Montrer que cette condition est nécessaire en donnant une assignation qui donne des valeurs différentes aux deux formules $\forall x(P(x) \wedge Q(x))$ et $P(x) \wedge (\forall x Q(x))$.*

□

Chapitre 5

Base de la démonstration automatique

Sommaire

5.1 Méthode de Herbrand **94**
5.1.1 Domaine et base de Herbrand 94
5.1.2 Interprétation de Herbrand 94
5.1.3 Théorème de Herbrand 95
5.2 Skolémisation **97**
5.2.1 Algorithme de skolémisation d'une formule 97
5.2.2 Propriétés de la forme de Skolem 99
5.2.3 Forme clausale 101
5.3 Unification **102**
5.3.1 Unificateur 102
5.3.2 Algorithme d'unification 103
5.4 Résolution au premier ordre **105**
5.4.1 Trois règles pour la résolution 105
5.4.2 Cohérence de la résolution 107
5.4.3 Complétude de la résolution 108
5.5 Outil logiciel **110**
5.6 Exercices **111**

Il est important de préciser que la logique du premier ordre est *indécidable*, cela signifie qu'il n'existe pas d'algorithme pour *déterminer* si une formule est valide ou non valide. Ce résultat fut établi par Alonzo Church et Alan Türing en 1936 et 1937 [4, 24]. Ils ont aussi montré que cette logique est *semi-décidable*, c'est-à-dire que nous pouvons écrire un programme, qui prend en entrée une formule et qui a le comportement suivant :

1. S'il termine alors il *décide correctement* si la formule est valide ou non. Lorsque la formule est valide, la décision est généralement accompagnée d'une preuve.
2. Si la formule est valide, alors il termine. Cependant, l'exécution peut être longue !

Notons que *si la formule n'est pas valide, la terminaison de ce programme n'est pas garantie*. En effet si le programme terminait pour toute formule, alors, d'après le premier point, ce serait un algorithme pour déterminer si une formule est valide ou non. Or, ainsi qu'il est dit ci-dessus, Church et Türing ont prouvé qu'un tel algorithme n'existait pas.

Plan : Dans ce chapitre nous commençons par présenter la méthode, introduite par Jacques Herbrand, permettant d'énumérer un ensemble fini de formules afin de trouver un modèle à une formule. Ensuite nous présentons la transformation due à Thoralf Albert Skolem qui transforme (skolémise) une formule en une « forme de Skolem » où l'existence d'un modèle est préservée. En transformant ensuite cette forme de Skolem en une « forme clausale » équivalente, nous pouvons appliquer la résolution. Nous reprenons alors le concept de résolution introduit dans le chapitre 2 page 37 et nous l'étudions en logique du premier ordre. En 1929, Kurt Gödel (1906 - 1978) prouva que la logique du premier ordre est complète et cohérente. Nous présentons ce résultat pour la résolution, c'est-à-dire qu'il existe une preuve par résolution en logique du premier ordre d'une formule F à partir d'un ensemble de formules Γ si et seulement si F est conséquence de Γ.

5.1 Méthode de Herbrand

Jacques Herbrand (1908-1931) est un mathématicien et logicien français. Il a établi en 1930 un lien entre calcul des prédicats et calcul des propositions.

5.1.1 Domaine et base de Herbrand

Nous introduisons d'abord la notion de fermeture universelle qui permet de transformer toute formule en une formule fermée où la propriété de satisfiabilité est maintenue.

Définition 5.1.1 (Fermeture universelle) *Soit C une formule ayant pour variables libres $x_1, \ldots, x_n$. La* fermeture universelle *de C, notée $\forall(C)$, est la formule $\forall x_1 \ldots \forall x_n C$. Cette notion est définie à l'ordre près des variables libres de C. Soit Γ un ensemble de formules, $\forall(\Gamma) = \{\forall(A) \mid A \in \Gamma\}$.*

Exemple 5.1.2 *La fermeture universelle de $P(x) \wedge R(x,y)$ est*

Nous généralisons la définition 1.3.1 page 19 aux termes de la logique du premier ordre.

Définition 5.1.3 (Substitution) *Une substitution est une application des variables dans les termes. Soient A une formule et σ une substitution. $A\sigma$ est la formule obtenue en remplaçant toute occurrence libre d'une variable par son image dans l'application. La formule $A\sigma$ est une* instance *de A.*

Dans la suite de ce chapitre, nous ne considérons que des formules qui ne contiennent ni le symbole égal, ni les constantes propositionnelles $\top$ et $\bot$, *car leur sens est fixé dans toute interprétation* : $\top$ vaut 1, $\bot$ vaut 0, et le symbole $=$ est l'identité sur le domaine de l'interprétation (voir Remarque 4.3.18 page 79). De plus nous considérons que toute signature comporte au moins une constante. Ainsi la signature d'un ensemble de formules ne comportant pas de constante, sera arbitrairement complétée par la constante a.

Définition 5.1.4 (Domaine et base de Herbrand)

1. *Le* domaine de Herbrand *pour Σ est l'ensemble des termes fermés (*i.e.*, sans variable) de cette signature, dénoté par D_Σ.*
2. *La* base de Herbrand *pour Σ est l'ensemble des formules atomiques fermées de cette signature, nous la notons B_Σ.*

Notons que le domaine de Herbrand d'une signature sans constante est vide. Puisque toutes les signatures considérées dans ce chapitre doivent comporter au moins une constante, ce domaine n'est pas vide.

Exemple 5.1.5 *Calculs de base de Herbrand :*

1. *Soit Σ la signature comportant uniquement les 2 constantes a, b et les 2 symboles de relations unaires P, Q. Nous avons $D_\Sigma = \{a, b\}$ et $B_\Sigma =$*
2. *Soit Σ la signature comportant uniquement la constante a, le symbole de fonction unaire f et le symbole de relation unaire P. Nous avons $D_\Sigma = \{f^n(a) \mid n \in \mathbb{N}\}$ et $B_\Sigma =$*

5.1.2 Interprétation de Herbrand

À partir de la base et du domaine de Herbrand d'une formule, nous sommes capables de construire une interprétation dite de « Herbrand ».

Définition 5.1.6 (Interprétation de Herbrand) *Soient Σ une signature et $E \subseteq B_\Sigma$. L'*interprétation de Herbrand *$H_{\Sigma,E}$ a pour domaine D_Σ et donne aux symboles le sens suivant :*

1. *Si le symbole s est une constante de la signature, il vaut lui-même dans cette interprétation.*
2. *Si s est un symbole de fonction à $n \geq 1$ arguments de la signature et si $t_1, \ldots, t_n \in D_\Sigma$ alors $s^{fn}_{H_{\Sigma,E}}(t_1, \ldots, t_n) = s(t_1, \ldots, t_n)$.*
3. *Si le symbole s est une variable propositionnelle, il vaut 1, autrement dit il est vrai, si et seulement si $s \in E$.*
4. *Si s est un symbole de relation de la signature à $n \geq 1$ arguments et si $t_1, \ldots, t_n \in D_\Sigma$ alors $s^{rn}_{H_{\Sigma,E}} = \{(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in D_\Sigma \wedge s(t_1, \ldots, t_n) \in E\}$.*

Le résultat suivant montre qu'une interprétation de Herbrand peut être identifiée à l'ensemble des formules atomiques fermées dont elle est modèle.

Propriété 5.1.7 (Propriété d'une interprétation de Herbrand) *Soient Σ une signature et $E \subseteq B_\Sigma$. Dans l'interprétation de Herbrand $H_{\Sigma,E}$:*

1. *La valeur d'un terme sans variable est lui-même.*
2. *L'interprétation est modèle d'une formule atomique fermée si et seulement si elle est élément de E.*

La preuve est une conséquence directe de la définition d'une interprétation de Herbrand.
Notons ici, avec un exemple, pourquoi on a supposé que les formules ne contiennent pas les symboles de relation $\top, \bot, =$, dont le sens est fixé dans toutes les interprétations. Supposons au contraire que $\top$ soit élément de la base mais non élément de E. D'après le point 2, l'interprétation $H_{\Sigma,E}$ donnerait $\top$ la valeur 0, alors que $\top$ vaut 1 dans toute interprétation.

Exemple 5.1.8 *Soit Σ la signature comportant les 2 constantes a, b et les 2 symboles de relations unaires P, Q. L'ensemble $E = \{P(b), Q(a)\}$ définit l'interprétation de Herbrand $H_{\Sigma,E}$ de domaine $D_\Sigma = \{a, b\}$*

Théorème 5.1.16 (Fermeture universelle et modèle de Herbrand) *Soit Γ un ensemble de formules sans quantificateur sur la signature Σ. La fermeture universelle $\forall(\Gamma)$ a un modèle si et seulement si $\forall(\Gamma)$ a un modèle qui est une interprétation de Herbrand de Σ.*

Ce théorème est appliqué dans la preuve du théorème 5.2.17 page 100.

5.1.3 Théorème de Herbrand

Le théorème de Herbrand établit un lien entre la logique du premier ordre et la logique propositionnelle.

Théorème 5.1.17 (Théorème de Herbrand) *Soit Γ un ensemble de formules sans quantificateur de signature Σ. $\forall(\Gamma)$ a un modèle si et seulement si tout ensemble fini d'instances fermées sur la signature Σ des formules de Γ a un modèle propositionnel application de la base de Herbrand B_Σ dans l'ensemble $\{0, 1\}$.*

Corollaire 5.1.18 *Soit Γ un ensemble de formules sans quantificateur de signature Σ. La fermeture universelle $\forall(\Gamma)$ est insatisfaisable si et seulement s'il existe un ensemble fini insatisfaisable d'instances fermées sur la signature Σ des formules de Γ.*

Preuve : Le corollaire est obtenu en remplaçant chaque côté de l'équivalence du théorème de Herbrand par sa négation. □

Dans le cas où Γ est un ensemble *fini* de formules, ce corollaire fonde une procédure de *semi-décision* pour savoir si $\forall(\Gamma)$ est ou n'est pas insatisfaisable : nous énumérons l'ensemble des instances fermées des formules de Γ sur la signature Σ et nous arrêtons cette énumération dès que nous obtenons un ensemble insatisfaisable, ou dès que cette énumération est terminée sans contradiction ou dès que nous sommes « fatigués » :

1. Dans le premier cas, la procédure répond (en appliquant le corollaire) que $\forall(\Gamma)$ est insatisfaisable.
2. Le deuxième cas ne peut se produire que si le domaine de Herbrand ne comprend que des constantes, la procédure répond (en appliquant le corollaire) que $\forall(\Gamma)$ est satisfaisable et en donne un modèle.
3. Dans le troisième cas, nous ne pouvons évidemment pas conclure : le corollaire nous dit que si $\forall(\Gamma)$ est insatisfaisable, et si nous avions été plus courageux, nous aurions obtenu une contradiction.

Exemple 5.1.19

1. *Soit* $\Gamma = \{P(x), Q(x), \neg P(a) \vee \neg Q(b)\}$. *La signature* Σ *comporte 2 constantes* a, b *et 2 symboles de relation unaire* P, Q.

2. *Soit* $\Gamma = \{P(x) \vee Q(x), \neg P(a), \neg Q(b)\}$. *Nous avons la même signature que précédemment.*

3. *Soit* $\Gamma = \{P(x), \neg P(f(x))\}$. *La signature* Σ *comporte une constante a, un symbole de fonction unaire* f *et un symbole de relation unaire P. La constante a est ajoutée à la signature de* Γ *pour que le domaine de Herbrand ne soit pas vide.*

4. *Soit* $\Gamma = \{P(x) \vee \neg P(f(x)), \neg P(a), P(f(f(a)))\}$. *Nous avons la même signature que précédemment.*

5. *Soit* $\Gamma = \{R(x, s(x)), R(x, y) \wedge R(y, z) \Rightarrow R(x, z), \neg R(x, x)\}$. *La signature* Σ *comporte une constante a, un symbole de fonction unaire s et un symbole de relation binaire R.*

5.2 Skolémisation

Le théorème de Herbrand est applicable à la fermeture universelle d'un ensemble de formules sans quantificateur. Nous étudions une transformation, la *skolémisation*, qui change un ensemble de formules fermées en la fermeture universelle d'un ensemble de formules sans quantificateur et *qui préserve l'existence d'un modèle.* Cette transformation est due à Thoralf Albert Skolem (1887 - 1963), mathématicien et logicien norvégien. Elle est nécessaire pour pouvoir appliquer la résolution sur un ensemble de formules de la logique du premier ordre. Nous commençons par donner l'intuition de la skolémisation avant de regarder le résultat de cette transformation sur des exemples simples afin d'observer les propriétés de la skolémisation. La skolémisation sert à éliminer les quantificateurs existentiels et change une formule fermée A en une formule B telle que :

— B a pour conséquence A.

— tout modèle de A donne un modèle de B.

Par suite A a un modèle si et seulement si B a un modèle : la skolémisation préserve l'existence d'un modèle, nous disons aussi qu'elle préserve la satisfaisabilité. Regardons ce qu'il en est sur deux exemples simples.

Exemple 5.2.1 *La formule $\exists x P(x)$ est skolémisée en $P(a)$. Nous observons les relations entre ces deux formules :*

1. *$P(a)$ a pour conséquence $\exists x P(x)$.*
2. *$\exists x P(x)$ n'a pas pour conséquence $P(a)$ mais un modèle de $\exists x P(x)$ « donne » un modèle de $P(a)$. En effet soit I un modèle de $\exists x P(x)$. Donc il existe $d \in P_I$. Soit J l'interprétation telle que $P_J = P_I$ et $a_J = d$. J est modèle de $P(a)$.*

Exemple 5.2.2 *La formule $\forall x \exists y Q(x,y)$ est skolémisée en $\forall x Q(x, f(x))$. Nous observons les relations entre ces deux formules :*

1. *$\forall x Q(x, f(x))$ a pour conséquence $\forall x \exists y Q(x,y)$.*
2. *$\forall x \exists y Q(x,y)$ n'a pas pour conséquence $\forall x Q(x, f(x))$ mais un modèle de $\forall x \exists y Q(x,y)$ « donne » un modèle de $\forall x Q(x, f(x))$. En effet soient I un modèle de $\forall x \exists y Q(x,y)$ et D le domaine de I. Pour tout $d \in D$, l'ensemble $\{e \in D \mid (d,e) \in Q_I\}$ n'est pas vide, donc il existe $g : D \to D$ une fonction[1] telle que pour tout $d \in D$, $g(d) \in \{e \in D \mid (d,e) \in Q_I\}$. Soit J l'interprétation telle que $Q_J = Q_I$ et $f_J = g$: J est modèle de $\forall x Q(x, f(x))$.*

5.2.1 Algorithme de skolémisation d'une formule

Nous définissons d'abord les éléments nécessaires pour présenter la forme de Skolem, ensuite nous donnons un algorithme de skolémisation.

Définition 5.2.3 (Formule propre) *Une formule fermée est dite* propre*, s'il elle ne comporte pas de variable liée par deux quantificateurs distincts.*

Exemple 5.2.4 *Les formules $\forall x P(x) \vee \forall x Q(x)$ et $\forall x (P(x) \Rightarrow \exists x Q(x) \wedge \exists y R(x,y))$ ne sont pas propres.*
Les formules $\forall x P(x) \vee \forall y Q(y)$ et $\forall x (P(x) \Rightarrow \exists y R(x,y))$ sont propres.

Nous étendons la notion de forme normale de la logique des propositions à la logique du premier ordre en disant qu'une formule est en forme normale si elle est sans équivalence, ni implication, et dont les négations portent uniquement sur les formules atomiques.

Définition 5.2.5 (Forme de Skolem) *Soient A une formule fermée et B la formule en forme normale sans quantificateur obtenue par la transformation ci-après : B est la* forme de Skolem *de A.*

Il y a plusieurs façons de skolémiser une formule fermée. Nous proposons un compromis entre l'efficacité et la simplicité de la transformation. Notre algorithme est consistué de la succession des quatre étapes suivantes :

1. **Transformation en formule normale** : Transformation de la formule fermée en une autre formule fermée en forme normale équivalente.
2. **Transformation en formule propre** : Transformation de la formule fermée en forme normale en une formule fermée, en forme normale, propre et équivalente.
3. **Élimination des quantificateurs existentiels** : Cette transformation préserve *seulement l'existence de modèle*, comme nous l'avons vu sur les exemples.
4. **Transformation en forme de Skolem** : Transformation de la formule fermée, en forme normale, propre et sans quantificateur existentiel en une formule en forme normale sans quantificateur.

Nous détaillons chacune de ces quatre étapes.

1. En affirmant l'existence de g, nous utilisons implicitement l'axiome du choix.

5.2.1.1 Transformation en formule normale

Comme dans le cas propositionnel, nous enlevons les équivalences et les implications et déplaçons les négations vers les formules atomiques au moyen des équivalences suivantes utilisées de gauche à droite :

— $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$.
— $A \Rightarrow B \equiv \neg A \vee B$.
— $\neg\neg A \equiv A$.
— $\neg(A \wedge B) \equiv \neg A \vee \neg B$.
— $\neg(A \vee B) \equiv \neg A \wedge \neg B$.
— $\neg\forall x A \equiv \exists x \neg A$.
— $\neg\exists x A \equiv \forall x \neg A$.

Remarque 5.2.6 *En combinant déplacement de la négation et élimination de l'implication, c'est-à-dire en remplaçant $\neg(A \Rightarrow B)$ par $A \wedge \neg B$, nous pouvons effectuer plus rapidement la transformation en formule normale équivalente, qu'en se limitant aux équivalences ci-dessus.*

Exemple 5.2.7 *La formule $\forall y(\forall x P(x,y) \Leftrightarrow Q(y))$ est transformée par élimination de l'équivalence et de l'implication en :*

5.2.1.2 Transformation en formule propre

En utilisant les résultats sur les changements de variables présentés dans la section 4.4.3 page 87, nous changeons le nom des variables liées à deux quantificateurs, en choisissant par exemple de nouvelles variables à chaque changement de nom.

Exemple 5.2.8 *La formule $\forall x P(x) \vee \forall x Q(x)$ est changée en*

La formule $\forall x(P(x) \Rightarrow \exists x Q(x) \wedge \exists y R(x,y))$ est changée en

5.2.1.3 Élimination des quantificateurs existentiels

Soit A une formule fermée normale et propre ayant une occurrence de la sous-formule $\exists y B$. Soient $x_1, \ldots, x_n$ l'ensemble des variables libres de $\exists y B$, où $n \geq 0$. Soit f un symbole *ne figurant pas dans A*. Nous remplaçons cette occurrence de $\exists y B$ par $B{<}y := f(x_1, \ldots, x_n){>}$[2].

Théorème 5.2.9 *Soit A' la formule obtenue en appliquant l'élimination d'un quantificateur existentiel sur la formule A. La formule A' est une formule fermée en forme normale et propre vérifiant :*

1. *A' a pour conséquence A ($A' \models A$).*
2. *Si A a un modèle alors A' a un modèle identique à celui de A sauf pour le sens de f.*

2. Si $n = 0$, f est une constante.

Remarque 5.2.10 *Dans le théorème 5.2.9 page précédente, il faut constater que la formule A' obtenue à partir de la formule A par élimination d'un quantificateur reste fermée, en forme normale et propre. Donc, en « appliquant » plusieurs fois le théorème, ce qui implique de choisir un* nouveau *symbole à chaque quantificateur éliminé, nous transformons une formule A fermée, en forme normale et propre en une formule B fermée, en forme normale, propre et* sans quantificateur existentiel *telle que :*

— *la formule A est conséquence de la formule B,*
— *si A a un modèle, alors B a un modèle identique sauf pour le sens des nouveaux symboles.*

Exemple 5.2.11 *En éliminant les quantificateurs existentiels de la formule $\exists x \forall y P(x,y) \wedge \exists z \forall u \neg P(z,u)$ nous obtenons*

Donc, il faut impérativement utiliser un nouveau symbole lors de chaque élimination d'un quantificateur existentiel.

Exemple 5.2.12 *En éliminant les quantificateurs existentiels de la formule $\exists x \forall y \exists z P(x,y,z)$ nous obtenons deux solutions possibles :*

— $\forall y P(a,y,f(a,y))$
— $\forall y P(a,y,f(y))$

Ceci dépend de l'ordre dans lequel nous appliquons l'elimination des quantificateurs existentiels. Or le théorème 5.2.9 montre que une des ces transformations a un modèle si et seulement si l'autre a également un modèle.

5.2.1.4 Transformation en fermeture universelle

Nous pouvons enfin enlever les quantificateurs universels afin d'obtenir une forme de Skolem.

Théorème 5.2.13 *Soit A une formule fermée, en forme normale, propre et sans quantificateur existentiel. Soit B la formule obtenue en enlevant de A tous les quantificateurs universels. La formule A est équivalente à la fermeture universelle de B.*

Preuve : Avec les conditions posées sur A, la transformation de A en $\forall(B)$ revient à effectuer tous les remplacements possibles des sous-formules de la forme suivante :

— $(\forall x C) \wedge D$ par $\forall x(C \wedge D)$, où x non libre dans D,
— $(\forall x C) \vee D$ par $\forall x(C \vee D)$, où x non libre dans D,
— $D \wedge (\forall x C)$ par $\forall x(D \wedge C)$, où x non libre dans D,
— $D \vee (\forall x C)$ par $\forall x(D \vee C)$, où x non libre dans D.

Puisque chacun de ces remplacements change une formule en une autre équivalente, les formules A et $\forall(B)$ sont équivalentes. □

5.2.2 Propriétés de la forme de Skolem

Propriété 5.2.14 *Soit A une formule fermée et B la forme de Skolem de A.*

— *La formule $\forall(B)$ a pour conséquence la formule A,*
— *si A a un modèle alors $\forall(B)$ a un modèle.*

Donc A a un modèle si et seulement si $\forall(B)$ a un modèle.

Preuve : Soit C la formule fermée en forme normale et propre, obtenue au terme des deux premières étapes de la skolémisation de A. Soit D le résultat de l'élimination des quantificateurs existentiels appliquée à C. D'après la remarque 5.2.10 nous avons :

— la formule D a pour conséquence la formule C,
— si C a un modèle alors D a un modèle.

Puisque les deux premières étapes changent des formules en des formules équivalentes, A et C sont équivalentes. D'après le théorème 5.2.13, D est équivalent à $\forall(B)$. Donc nous pouvons remplacer ci-dessus D par $\forall(B)$ et C par A, ce qui prouve le théorème. □

Exemple 5.2.15 *Soit $A = \forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$. Nous skolémisons $\neg A$.*

1. *$\neg A$ est transformée en la formule en forme normale :*

2. *La formule en forme normale est transformée en la formule propre :*

3. *Le quantificateur existentiel est « remplacé » par une constante :*

4. *Les quantificateurs universels sont enlevés :*

   Cette formule est la forme de Skolem de $\neg A$ suivant la définition 5.2.5 page 97.

Instancions la forme de Skolem de $\neg A$ en remplaçant x et y par a. Nous obtenons la formule $(\neg P(a) \vee Q(a)) \wedge P(a) \wedge \neg Q(a)$ qui est insatisfaisable. Donc $\forall((\neg P(x) \vee Q(x)) \wedge P(y) \wedge \neg Q(a))$ est insatisfaisable. Puisque, d'après la propriété 5.2.14 page précédente, la skolémisation préserve l'existence d'un modèle, $\neg A$ est insatisfaisable, donc A est valide.

Pour skolémiser un ensemble de formules, nous skolémisons chaque formule de l'ensemble comme il a été indiqué ci-dessus en choisissant impérativement un nouveau symbole pour chaque quantificateur existentiel éliminé durant la troisième étape de la skolémisation. Les raisons pour le choix d'un symbole nouveau apparaissent clairement dans la preuve du théorème 5.2.9 page 98 et dans l'exemple 5.2.11 page précédente.

Corollaire 5.2.16 *En skolémisant un ensemble de clauses fermées Γ, nous obtenons un ensemble Δ de formules sans quantificateurs tel que :*
- *Tout modèle de $\forall(\Delta)$ est modèle de Γ.*
- *Si Γ a un modèle alors $\forall(\Delta)$ en a un modèle qui ne diffère de celui de Γ que par le sens des nouveaux symboles.*

La skolémisation a des conséquences pratiques : elle facilite la preuve d'insatisfiabilité d'une formule. Elle a aussi des conséquences théoriques, que nous examinons.

Théorème 5.2.17 *Soit Γ un ensemble dénombrable de formules fermées. Si Γ a un modèle alors Γ a un modèle dénombrable.*

Preuve : Nous skolémisons Γ en un ensemble Δ de formules sans quantificateurs. Supposons que Γ a un modèle. Alors $\forall(\Delta)$ a un modèle d'après le corollaire 5.2.16. D'après le théorème 5.1.16 page 95, $\forall(\Delta)$ a un modèle de Herbrand sur la signature de Δ. Puisque l'ensemble Γ est dénombrable, sa signature l'est aussi, donc aussi celle de Δ et par suite le domaine de ce modèle de Herbrand est dénombrable. □

Remarque 5.2.18 *Appelons* théorie, *un ensemble de formules. D'après le théorème, toute théorie dénombrable qui a un modèle, a un modèle dénombrable. Autrement dit avec une théorie du premier ordre, nous pouvons parler des propriétés des nombres réels ou des ensembles, mais nous ne pouvons pas obliger notre théorie à n'avoir que des modèles non dénombrables, bien que l'ensemble des réels ou la classe de tous les ensembles ne soient pas dénombrables.*

5.2.3 Forme clausale

Après la skolémisation des formules et avant de pouvoir appliquer une généralisation au premier ordre de la méthode de résolution il faut transformer les formes de Skolem en clauses comme nous l'avons fait en logique propositionnelle. Nous étendons donc les notions essentielles à cette transformation avant de donner les règles permettant cette transformation.

Définition 5.2.19 (Littéral positif, négatif et clause) *Un* littéral positif *est une formule atomique. Un* littéral négatif *est la négation d'une formule atomique.*

Nous remarquons que tout littéral est positif ou négatif, et nous rappelons qu'une clause est une somme de littéraux.

Définition 5.2.20 (Forme clausale d'une formule) *Soit A une formule fermée, la* forme clausale *de A est un ensemble de clauses obtenu en deux étapes :*

1. *Skolémisation de A, autrement dire construire B la forme de Skolem de A.*
2. *Remplacement de B par un ensemble Γ équivalent de clauses obtenu par distributivité de la somme sur le produit.*

Propriété 5.2.21 *La fermeture universelle de la forme clausale d'une formule fermée a un modèle si et seulement la formule a un modèle. Plus précisément :*
- *la fermeture universelle de la forme clausale d'une formule a pour conséquence la formule,*
- *si la formule a un modèle alors la fermeture universelle de sa forme clausale en a un.*

Preuve : Soient A une formule fermée, B sa forme de Skolem et Γ sa forme clausale. D'après les propriétés de la skolémisation :
- $\forall(B)$ a pour conséquence A.
- Si A a un modèle alors $\forall(B)$ a un modèle.

Puisque Γ est obtenu par distributivité, B et Γ sont équivalents donc $\forall(B)$ et $\forall(\Gamma)$ sont aussi équivalents. Par suite dans les deux propriétés ci-dessus, nous pouvons remplacer $\forall(B)$ par $\forall(\Gamma)$. □

Définition 5.2.22 (Forme clausale d'un ensemble de formules) *Soit Γ un ensemble de formules fermées. Nous définissons la* forme clausale *de Γ comme l'union des formes clausales de chacune des formules de Γ, en prenant soin au cours de la skolémisation d'éliminer chaque occurrence d'un quantificateur existentiel à l'aide d'un nouveau symbole.*

Corollaire 5.2.23 *Soient Γ un ensemble de formules fermées et Δ la forme clausale de Γ. Nous avons :*
- *$\forall(\Delta)$ a pour conséquence Γ, et*
- *si Γ a un modèle alors $\forall(\Delta)$ a un modèle.*

Théorème 5.2.24 (Adaptation du théorème de Herbrand aux formes clausales) *Soient Γ un ensemble de formules fermées et Δ la forme clausale de Γ. L'ensemble Γ est insatisfaisable si et seulement s'il existe un sous-ensemble fini insatisfaisable d'instances des clauses de Δ sur la signature de Δ.*

Preuve : D'après l5.2.23, la skolémisation préserve la satisfaisablité, donc : Γ est insatisfaisable si et seulement si $\forall(\Delta)$ est insatisfaisable. D'après le corollaire du théorème de Herbrand 5.1.18 page 95, $\forall(\Delta)$ est insatisfaisable si et seulement s'il existe un sous-ensemble fini insatisfaisable d'instances des clauses de Δ sur la signature de Δ. □

Exemple 5.2.25 *Soit $A = \exists y \forall z (P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z)))$. Calculons la forme clausale de A.*

1. *Mettre A sous forme en forme normale :*

2. *Rendre propre le résultat :*

3. *Éliminer les quantificateurs existentiels :*

4. *Supprimer les quantificateurs universels, nous obtenons la forme de Skolem de A :*

5. *Transformer en produit de sommes de littéraux, nous obtenons la forme clausale de A, qui est l'ensemble suivant de clauses :*

D'après la propriété ci-dessus, A n'a pas de modèle si et seulement s'il y a un ensemble fini insatisfaisable d'instances de C_1, C_2, C_3 *sur la signature de ces clauses. Nous recherchons ces instances :*

5.3 Unification

Après la mise en forme clausale d'un ensemble de formules en logique du premier ordre et avant de généraliser la résolution pour ces clauses, nous avons besoin d'un algorithme résolvant l'égalité entre deux termes. Étant donné deux termes, trouver une substitution qui appliquée à ces deux termes les rendent égaux est appelé *problème d'unification* de ces deux termes. Ce problème a intéressé de grands noms de l'informatique comme Alan Robinson [23], Gérard Huet [15, 16], Alberto Martelli et Ugo Montanari [21], ou encore plus récemment Franz Baader et Ayne Snyder [3].

5.3.1 Unificateur

Définition 5.3.1 (Expression, solution) *Appelons* expression*, un terme ou un littéral. Une substitution* σ *(voir définition 5.1.3 page 94) est* solution de l'équation $e_1 = e_2$ *entre deux expressions, si les deux expressions* $e_1\sigma$ *et* $e_2\sigma$ *sont* identiques*. Une substitution est solution d'un ensemble d'équations si elle est solution de chaque équation de l'ensemble.*

Définition 5.3.2 (Unificateur) *Soient* σ *une substitution et E un ensemble d'expressions.* $E\sigma = \{t\sigma \mid t \in E\}$*. La substitution* σ *est un unificateur de E si et seulement si l'ensemble* $E\sigma$ *n'a qu'un élément. Soit* $\{e_i | 1 \leq i \leq n\}$ *un ensemble fini d'expressions. La substitution* σ *est un unificateur de cet ensemble si et seulement si elle est solution du système d'équations* $\{e_i = e_{i+1} | 1 \leq i < n\}$*.*

Définition 5.3.3 (Support d'une substitution) *Le* support *d'une substitution* σ *est l'ensemble des variables x telles que* $x\sigma \neq x$*. Dans la suite, nous nous intéressons seulement aux substitutions à support fini, c'est à dire qui ne changent qu'un nombre fini de variables. Une substitution* σ *à support fini est notée* $< x_1 := t_1, \ldots, x_n := t_n >$ *ou plus simplement* $x_1 := t_1, \ldots, x_n := t_n$ *quand il n'y a pas de risque d'ambiguïté. Les variables* $x_1, \ldots, x_n$ *sont distinctes et la substitution vérifie :*

— *Pour i de* 1 *à n,* $x_i\sigma = t_i$.
— *Pour toute variable y telle que* $y \notin \{x_1, \ldots, x_n\}$*, nous avons :* $y\sigma = y$.

Exemple 5.3.4 *L'équation* $P(x, f(y)) = P(g(z), z)$ *a pour solution :*

Le système d'équations $x = g(z), f(y) = z$ *a pour solution :*

Définition 5.3.5 (Composition de substitutions) *Soient* σ *et* τ *deux substitutions, nous notons* $\sigma\tau$ *la substitution telle que pour toute variable x,* $x\sigma\tau = (x\sigma)\tau$*. La substitution* $\sigma\tau$ *est une instance de* σ*. Deux substitutions sont équivalentes si chacune d'elles est une instance de l'autre.*

Exemple 5.3.6 *Considérons les substitutions suivantes :*
— $\sigma_1 =< x := g(z), y := z >$.
— $\sigma_2 =< x := g(y), z := y >$.
— $\sigma_3 =< x := g(a), y := a, z := a >$.
Nous avons les relations suivantes entre ces substitutions :

Définition 5.3.7 (Solution la plus générale) *Une solution d'un système d'équations est appelée* la plus générale *(en français nous disons aussi* principale*) si toute autre solution en est une instance. Notons que deux solutions « les plus générales » sont équivalentes.*

Exemple 5.3.8 *L'équation* $f(x, g(z)) = f(g(y), x)$ *a pour solutions (entre autres) :*

Définition 5.3.9 (Unificateur le plus général) *Soit E un ensemble d'expressions. Nous rappelons qu'une expression est un terme ou un littéral. Un unificateur de E (voir définition* *5.3.2 page précédente**) est appelé* le plus général *(ou encore principal), si tout autre unificateur en est une instance.*

Remarque 5.3.10 (Unificateur le plus général et solution la plus générale) *Soit* $E = \{e_i \mid 1 \leq i \leq n\}$ *un ensemble d'expressions. Dans la définition d'un unificateur, nous avons indiqué que* σ *est un unificateur de E si et seulement si* σ *est solution du système* $S = \{e_i = e_{i+1} \mid 1 \leq i < n\}$.
Donc l'unificateur le plus général de E est la solution la plus générale de S.

5.3.2 Algorithme d'unification

L'algorithme calculant la solution d'un système d'équations est appelé *algorithme d'unification* car la recherche d'un unificateur d'un ensemble d'expressions se réduit à la recherche d'une solution d'un système d'équations. L'algorithme sépare les équations en équations à résoudre, notées par une égalité et équations résolues, notées par le signe :=. Initialement, il n'y a pas d'équations résolues. L'algorithme applique les règles énoncées ci-dessous. Il s'arrête quand il n'y a plus d'équations à résoudre ou quand il a déclaré que le système à résoudre n'a pas de solution. Quand il s'arrête sans avoir déclaré l'absence de solution, la liste des équations résolues est la solution la plus générale du système initial d'équations.

5.3.2.1 Les règles de l'algorithme

Notre algorithme d'unification consiste à appliquer les six règles suivantes :

1. **Supprimer.** Si les deux membres d'une équation sont identiques, l'équation est supprimée[3].
2. **Décomposer.** Si les 2 membres d'une équation sont distincts et commencent par le même symbole alors :
   — Une équation de la forme $\neg A = \neg B$ est remplacée par $A = B$.
   — Une équation de la forme $f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)$ est remplacée par les équations $s_1 = t_1,\ldots,s_n = t_n$. Pour $n = 0$, cette décomposition supprime l'équation.
3. **Échec de la décomposition.** Si une équation à résoudre est de la forme $f(s_1,\ldots,s_n) = g(t_1,\ldots,t_p)$ avec $f \neq g$ ou $n \neq p$ alors l'algorithme déclare qu'il n'y a pas de solution. En particulier il y a évidemment un échec si nous cherchons à résoudre une équation entre un littéral positif et un littéral négatif ou bien si nous avons une constante et une fonction.
4. **Orienter.** Si une équation est de la forme $t = x$ où t est un terme qui n'est pas une variable et x une variable, alors nous remplaçons l'équation par $x = t$.
5. **Élimination d'une variable** : Si une équation à résoudre est de la forme $x = t$ où x est une variable et t un terme *ne contenant pas* x alors :
   (a) Enlever des équations à résoudre $x = t$.
   (b) Remplacer x par t dans toutes les équations (non résolues *et résolues*)[4].
   (c) Ajouter $x := t$ à la partie résolue.
6. **Échec de l'élimination** : Si une équation à résoudre est de la forme $x = t$ où x est une variable et t un terme distinct de x et *contenant* x alors l'algorithme déclare qu'il n'y a pas de solution.

Exemple 5.3.11 *Nous appliquons cet algorithme afin de résoudre les équations suivantes :*

1. $f(x,g(z)) = f(g(y),x)$.

2. $f(x,x,a) = f(g(y),g(a),y)$.

3. $f(x,x,x) = f(g(y),g(a),y)$.

3. Nous pouvons nous limiter à la suppression des équations de la forme $x = x$ où x est une variable.
4. Un algorithme efficace évite ce remplacement systématique.

Nous montrons d'abord que notre algorithme est correct, c'est-à-dire qu'il calcule bien un unificateur. Ensuite nous démontrons qu'il termine toujours.

5.3.2.2 Correction de l'algorithme

Nous observons facilement les trois faits suivants :

— (1) : les règles de suppression, de décomposition, d'orientation et d'élimination des variables préservent l'ensemble des solutions.

— (2) : initialement et après toute application de règle, l'ensemble des équations résolues forme une substitution $x_1 := t_1, \ldots, x_n := t_n$ où $x_1, \ldots, x_n$ sont des variables distinctes ne figurant pas dans les termes $t_1, \ldots, t_n$. Soit σ cette substitution. Il est clair que $t_i\sigma = t_i$ ainsi σ est solution des équations résolues.

— (3) : Les règles d'échec déclarent correctement l'absence de solution.

De (1) et (2), il résulte que si l'algorithme termine sans échec, la substitution σ trouvée est la solution la plus générale du *système initial*. En effet :

— σ est une solution, puisque, d'après (2), elle est solution du *système terminal* et que, d'après (1), l'ensemble des solutions est préservé par les règles.

— σ est une solution la plus générale, puisque, d'après (1), toute autre solution est une solution du système terminal, donc une instance de la substitution définie par ce système.

De (1) et de (3), il résulte que si l'algorithme déclare un échec, alors le système initial n'a pas de solution. En effet, dans ce cas :

— si le système initial avait une solution, d'après (1), elle serait solution du système terminal,

— d'après (3), le système terminal n'a pas de solution.

5.3.2.3 Terminaison de l'algorithme

— Les règles de suppression et de décomposition font décroître strictement la somme des longueurs des termes qui sont de part et d'autre des équations.

— La règle d'orientation fait décroître strictement la somme des longueurs des termes qui sont à gauche des équations.

— La règle d'élimination fait décroître strictement le nombre de variables des équations non résolues.

Il résulte que l'algorithme d'unification termine.

5.4 Résolution au premier ordre

Nous avons maintenant tous les ingrédients pour présenter la résolution au premier ordre. Bien entendu nous transformons un ensemble de formules en forme clausale. Ensuite nous pouvons appliquer trois règles permettant de « factoriser » des termes, de « copier » une clause et enfin de faire une « résolution binaire ». Ces différentes règles utilisent l'algorithme d'unification présenté précédemment. Par la suite, nous montrons la cohérence et la complétude de ce système formel. La complétude est basée sur le théorème de Herbrand dans la version donnée dans le corollaire 5.2.24 page 101.

5.4.1 Trois règles pour la résolution

Soit Γ un ensemble de clauses, nous supposons que $\forall(\Gamma)$ n'a pas de modèle, nous proposons alors un système formel permettant de déduire $\perp$ de Γ avec les trois règles suivantes :

1. La *factorisation* qui de la prémisse $P(x, f(y)) \vee P(g(z), z) \vee Q(z, x)$ déduit $P(g(f(y)), f(y)) \vee Q(f(y), g(f(y)))$. La clause déduite est obtenue en calculant la solution la plus générale $x := g(f(y)), z := f(y)$ de $P(x, f(y)) = P(g(z), z)$.
2. La règle de *copie* qui permet de renommer les variables d'une clause.
3. La *résolution binaire* (RB) qui des deux prémisses sans variable commune $P(x, a) \vee Q(x)$ et $\neg P(b, y) \vee R(f(y))$ déduit le résolvant $Q(b) \vee R(f(a))$, en calculant la solution plus générale $x := b, y := a$ de $P(x, a) = P(b, y)$.

Pour simplifier la présentation des règles, nous identifierons une clause, qui est une somme de littéraux, avec l'ensemble de ses littéraux. Nous formalisons ces trois règles.

5.4.1.1 Factorisation

Propriété 5.4.1 *Soient A une formule sans quantificateur et B une instance de A. Nous avons :* $\forall(A) \models \forall(B)$.

Définition 5.4.2 *La clause C' est un* facteur *de la clause C si $C' = C$ ou s'il existe un sous-ensemble E de C tel que E a au moins deux éléments, E est unifiable et $C' = C\sigma$ où σ est l'unificateur le plus général de E.*

Exemple 5.4.3 *La clause $\underline{P(x)} \vee Q(g(x, y)) \vee \underline{P(f(a))}$ a deux facteurs :*

Nous obtenons immédiatement la propriété suivante.

Propriété 5.4.4 (Cohérence de la règle de factorisation) *Soit C' un facteur de la clause C. Nous avons :* $\forall(C) \models \forall(C')$.

Preuve : Puisque C' est une instance de C, la propriété est une conséquence immédiate de la propriété 5.4.1. □

5.4.1.2 Copie d'une clause

Définition 5.4.5 *Soient C une clause et σ une substitution, qui ne change que les variables de C et dont la* restriction *aux variables de C est une* bijection *entre ces variables et celles de la clause $C\sigma$. La clause $C\sigma$ est une copie de la clause C. La substitution σ est aussi appelée un* renommage *de C.*

Définition 5.4.6 *Soient C une clause et σ un renommage de C. Soit f la restriction de σ aux variables de C et f^{-1} l'application réciproque de f. Soit σ_C^{-1} la substitution ainsi définie pour toute variable x :*

— *Si x est une variable de $C\sigma$ alors $x\sigma_C^{-1} = xf^{-1}$ (par soucis de clarté, nous avons préféré utiliser la notation postfixée xf^{-1} plutôt que la notation préfixée $f^{-1}(x)$, plus usuelle).*
— *Sinon $x\sigma_C^{-1} = x$.*

*Cette substitution est appelée l'*inverse du renommage *σ de C.*

Exemple 5.4.7 *Soit $\sigma = <x := u, y := v>$. σ est un renommage du littéral $P(x, y)$. Le littéral $P(u, v)$, où $P(u, v) = P(x, y)\sigma$, est une copie de $P(x, y)$. Soit $\tau = <u := x, v := y>$. τ est l'inverse du renommage σ de $P(x, y)$. Notons que $P(u, v)\tau = P(x, y)$: le littéral $P(x, y)$ est une copie de $P(u, v)$ par le renommage τ.*

Propriété 5.4.8 *Soient C une clause et σ un renommage de C.*

1. *σ_C^{-1} est un renommage de $C\sigma$.*
2. *Pour toute expression ou clause E, dont les variables sont celles de C, $E\sigma\sigma_C^{-1} = E$.*

Donc $C\sigma\sigma_C^{-1} = C$ et par suite C est une copie de $C\sigma$.

Preuve : Soit f la restriction de σ aux variables de C. Par définition du renommage, f est une bijection entre les variables de C et celles de $C\sigma$.

1. Par définition de σ_C^{-1}, cette substitution ne change que les variables de $C\sigma$ et sa restriction aux variables de $C\sigma$ est la bijection f^{-1}. Donc, σ_C^{-1} est un renommage de $C\sigma$.

2. Soit x une variable de C. Par définition de f, $x\sigma\sigma_C^{-1} = xff^{-1} = x$. Donc, par une récurrence omise et sans intérêt sur les termes, littéraux et clauses, pour toute expression ou clause E, dont les variables sont celles de C, nous avons $E\sigma\sigma_C^{-1} = E$.

□

Propriété 5.4.9 *Soient deux clauses copies l'une de l'autre, leurs fermetures universelles sont équivalentes.*

Preuve : Soit C' une copie de C. Par définition, C' est une instance de C et par la propriété précédente, C est une copie de C', donc une instance de C'. Donc par la propriété 5.4.1 page ci-contre, la fermeture universelle de C est conséquence de celle de C' et inversement. Par suite, ces deux fermetures universelles sont équivalentes. □

5.4.1.3 Résolution binaire

Définition 5.4.10 *Soient C et D deux clauses* n'ayant pas de variable commune. *La clause E est un* résolvant binaire *de C et D s'il y a un littéral $L \in C$ et un littéral $M \in D$ tels que L et M^c (le littéral opposé à M) sont unifiables et si $E = ((C - \{L\}) \cup (D - \{M\}))\sigma$ où σ est la solution la plus générale de l'équation $L = M^c$.*

Exemple 5.4.11 *Soient $C = P(x,y) \vee P(y,k(z))$ et $D = \neg P(a, f(a,y_1))$.*

Propriété 5.4.12 (Cohérence de la règle de résolution binaire) *Soit E un résolvant binaire des clauses C et D, nous avons : $\forall(C), \forall(D) \models \forall(E)$.*

Preuve : Soit I une interprétation. Puisque les trois formules $\forall(C), \forall(D), \forall(E)$ sont sans variable libre, il suffit de montrer que si I est modèle de $\forall(C)$ et de $\forall(D)$ alors I est modèle de $\forall(E)$. Supposons que I est modèle de $\forall(C), \forall(D)$ et montrons que c'est un modèle de $\forall(E)$. Soit e un état *quelconque* de cette interprétation. (I,e) est modèle de $C\sigma$ et $D\sigma$, pour toute substitution σ. Par définition du résolvant binaire, il y a un littéral $L \in C$ et un littéral $M \in D$ tels que L et M^c (le littéral opposé à M) sont unifiables. Ainsi, $L\sigma$ et $M\sigma$ sont deux littéraux opposés, donc (I,e) est modèle soit de l'un, soit de l'autre de ces deux littéraux.

1. Supposons (I,e) modèle de $L\sigma$. Puisque (I,e) est contre-modèle de $M\sigma$ et modèle de $D\sigma$, c'est un modèle de $(D - \{M\})\sigma$ donc de E car $(D - \{M\})\sigma \subset E$.
2. Supposons (I,e) contre-modèle de $L\sigma$. Puisque c'est un modèle de $C\sigma$, c'est un modèle de $(C - \{L\})\sigma$ donc de E car $(C - \{L\})\sigma \subset E$.

Donc (I,e) est modèle de E. Puisque l'état e est quelconque, I est modèle de $\forall(E)$. □

5.4.1.4 Preuve par factorisation, copie et résolution binaire

Définition 5.4.13 *Soient Γ un ensemble de clauses et C une clause. Une* preuve *de C à partir de Γ est une suite de clauses se terminant par C, toute clause de la preuve étant un élément de Γ, un facteur d'une clause la précédant dans la preuve, une copie d'une clause la précédant dans la preuve ou un résolvant binaire de deux clauses la précédant dans la preuve.*

C est déduite de Γ au premier ordre par les 3 règles de factorisation, copie et résolution binaire est dénoté par $\Gamma \vdash_{1fcb} C$. Cela signifie qu'il y a une preuve de C à partir de Γ. Quand il n'y a pas d'ambiguïté sur le système formel utilisé, nous remplaçons $\vdash_{1fcb}$ par $\vdash$.

5.4.2 Cohérence de la résolution

Propriété 5.4.14 (Cohérence) *Soient Γ un ensemble de clauses et C une clause. Si $\Gamma \vdash_{1fcb} C$ alors $\forall(\Gamma) \models \forall(C)$.*

Preuve : Cette propriété est une conséquence immédiate de la cohérence de la factorisation, de la copie et de la résolution binaire. Cette preuve est demandée dans l'exercice 97 page 114. □

Exemple 5.4.15 *Soient les deux clauses :*

1. $C_1 = P(x,y) \vee P(y,x)$.
2. $C_2 = \neg P(u,z) \vee \neg P(z,u)$.

Montrons par résolution que $\forall(C_1, C_2)$ n'a pas de modèle.

Cet exemple montre, a contrario, que la résolution binaire seule est incomplète, sans la factorisation, nous ne pouvons pas déduire la clause vide.

Exemple 5.4.16 *Soient les trois clauses :*

1. $C_1 = \neg P(z,a) \vee \neg P(z,x) \vee \neg P(x,z)$.
2. $C_2 = P(z, f(z)) \vee P(z,a)$.
3. $C_3 = P(f(z), z) \vee P(z,a)$.

Nous donnons une preuve par résolution que $\forall(C_1, C_2, C_3)$ n'a pas de modèle. Dans cette preuve RB 1(3), 3(1) signifie par résolution binaire sur le 3º littéral de la clause 1 et le 1º littéral de la clause 3.

5.4.3 Complétude de la résolution

Nous définissons une *nouvelle* règle, la résolution au 1º ordre, qui est une combinaison des trois règles de factorisation, copie et résolution binaire. Ceci nous permettra de prouver la complétude de la résolution.

Définition 5.4.17 *La clause E est un résolvant au 1° ordre des clauses C et D si E est un résolvant binaire de C' et D' où C' est un facteur de C et D' est une copie sans variable commune avec C' d'un facteur de D. La règle qui de C et D déduit E est appelée la résolution de 1° ordre.*

Exemple 5.4.18 *Soient $C = \neg P(z,a) \vee \neg P(z,x) \vee \neg P(x,z)$ et $D = P(z, f(z)) \vee P(z,a)$.*

$C' = \neg P(a,a)$ est un facteur de C. La clause $P(a, f(a))$ est un résolvant binaire de C' et de D (qui est facteur de lui-même) donc

Soient Γ un ensemble de clauses et C une clause, nous avons défini jusqu'à présent trois notions de preuve par résolution que nous distinguons en notant par :

1. $\Gamma \vdash_p C$ le fait qu'il y existe une preuve de C à partir de Γ obtenue par résolution propositionnelle, autrement dit sans substitution.
2. $\Gamma \vdash_{1fcb} C$ le fait qu'il y existe une preuve de C à partir de Γ par factorisation, copie et résolution binaire.
3. $\Gamma \vdash_{1r} C$ le fait qu'il y existe une preuve de C à partir de Γ obtenue par résolution de 1° ordre.

Puisque la résolution du premier ordre est une combinaison des règles factorisation, copie et résolution binaire, nous avons immédiatement $\Gamma \vdash_{1r} C$ implique $\Gamma \vdash_{1fcb} C$.

Théorème 5.4.19 (Théorème du relèvement) *Soient C et D deux clauses. Soient C' une instance de C et D' une instance de D. Soit E' un résolvant* propositionnel *de C' et D', il existe E un résolvant au* premier ordre *de C et D qui a pour instance E'.*

Exemple 5.4.20 *Soient $C = P(x) \vee P(y) \vee R(y)$ et $D = \neg Q(x) \vee P(x) \vee \neg R(x) \vee P(y)$.*

— *Les clauses $C' = P(a) \vee R(a)$ et $D' = \neg Q(a) \vee P(a) \vee \neg R(a)$ sont des instances respectivement de C et D.*
— *La clause $E' = P(a) \vee \neg Q(a)$ est un résolvant propositionnel de C' et D'.*
— *La clause $E = P(x) \vee \neg Q(x)$ est un résolvant au 1° ordre de C et D qui a pour instance E'.*

Théorème 5.4.21 *Soient Γ un ensemble de clauses et Δ un ensemble d'instances des clauses de Γ, et $C_1, \ldots, C_n$ une preuve par résolution propositionnelle à partir de Δ, il existe une preuve $D_1, \ldots, D_n$ par résolution au 1° ordre à partir de Γ telle que pour i de 1 à n, la clause C_i est une instance de D_i.*

Preuve : Nous effectuons une récurrence sur n. Soit $C_1, \ldots, C_n, C_{n+1}$ une preuve par résolution propositionnelle à partir de Δ. Par récurrence, il existe une preuve $D_1, \ldots, D_n$ par résolution 1° ordre à partir Γ telle que pour i de 1 à n, la clause C_i est une instance de D_i. Nous distinguons deux cas :

1. Supposons $C_{n+1} \in \Delta$. Il existe $E \in \Gamma$ dont C_{n+1} est une instance donc nous prenons $D_{n+1} = E$.
2. Supposons que C_{n+1} soit un résolvant propositionnel de C_j et C_k où $j,k \leq n$. D'après le résultat précédent, il existe E résolvant au 1° ordre de D_j et D_k : nous prenons $D_{n+1} = E$.

□

Nous en déduisons immédiatement le corollaire suivant.

Corollaire 5.4.22 *Soient Γ un ensemble de clauses et Δ un ensemble d'instances des clauses de Γ. Supposons que $\Delta \vdash_p C$. Il existe D telle que $\Gamma \vdash_{1r} D$ et C est une instance de D.*

Exemple 5.4.23 *Soit l'ensemble de clauses $P(f(x)) \vee P(u), \neg P(x) \vee Q(z), \neg Q(x) \vee \neg Q(y)$. La fermeture universelle de cet ensemble de clauses est insatisfaisable et nous le montrons de trois manières.*

1. *Par instanciation sur le domaine de Herbrand $\{f^n(a), n \in \mathbb{N}\}$:*
   — *$P(f(x)) \vee P(u)$ est instanciée par $x := a, u := f(a)$ en $P(f(a))$*
   — *$\neg P(x) \vee Q(z)$ est instanciée par $x := f(a), z := a$ en $\neg P(f(a)) \vee Q(a)$*
   — *$\neg Q(x) \vee \neg Q(y)$ est instanciée par $x := a, y := a$ en $\neg Q(a)$*
   L'ensemble de ces 3 instanciations est insatisfaisable, comme le montre la preuve par résolution propositionnelle ci-dessous :

$$\dfrac{\dfrac{P(f(a)) \quad \neg P(f(a)) \vee Q(a)}{Q(a)} \qquad \neg Q(a)}{\bot}$$

2. *Cette preuve par résolution propositionnelle est relevée en une preuve par la règle de résolution au premier ordre :*

$$\dfrac{\dfrac{P(f(x)) \vee P(u) \quad \neg P(x) \vee Q(z)}{Q(z)} \qquad \neg Q(x) \vee \neg Q(y)}{\bot}$$

3. *Chaque règle de résolution au premier ordre est décomposée en factorisation, copie et résolution binaire :*

$$\dfrac{\dfrac{\dfrac{P(f(x)) \vee P(u)}{P(f(x))}\,Fact \quad \dfrac{\neg P(x) \vee Q(z)}{\neg P(y) \vee Q(z)}\,Copie}{Q(z)}\,RB \qquad \dfrac{\neg Q(x) \vee \neg Q(y)}{\neg Q(x)}\,Fact}{\bot}\,RB$$

Théorème 5.4.24 (Complétude réfutationnelle de la résolution au 1° ordre) *Soit Γ un ensemble de clauses. Les propositions suivantes sont équivalentes :*

1. $\Gamma \vdash_{1r} \bot$.
2. $\Gamma \vdash_{1fcb} \bot$.
3. $\forall(\Gamma) \models \bot$.

Preuve :

— Nous savons que (1) implique (2), car la résolution au 1° ordre est une combinaison de factorisation, copie et résolution binaire.

— Nous savons que (2) implique (3), car la factorisation, la copie et la résolution binaire sont cohérentes.

— Il nous reste à montrer que (3) implique (1). Supposons que $\forall(\Gamma) \models \bot$, autrement dit $\forall(\Gamma)$ est insatisfaisable. D'après le théorème de Herbrand, il y a Δ un ensemble fini d'instances sans variable de clauses de Γ qui n'a pas de modèle propositionnel. Par complétude de la résolution propositionnelle, nous avons : $\Delta \vdash_p \bot$. D'après le corollaire au relèvement 5.4.22 page précédente, il existe D telle que $\Gamma \vdash_{1r} D$ et $\bot$ est instance de D. Dans ce cas, nous avons $D = \bot$.

□

5.5 Outil logiciel

Nous indiquons ici un logiciel pour expérimenter le système de déduction décrit dans ce chapitre :

`http://teachinglogic.univ-grenoble-alpes.fr/ResBinSc/`

Cet outil, de façon similaire à celui proposé pour la déduction naturelle propositionnelle au paragraphe 3.5 page 66, permet de saisir une liste de clauses du premier ordre. Il tente alors automatiquement d'en déduire la clause vide :

— si cette liste de clauses est insatisfaisable et qu'aucune limite de temps ni de taille n'a été fixée, alors la clause vide sera déduite ;

— il est possible de restreindre l'espace de recherche, en imposant une limite de temps ou de taille sur les clauses produites (mais alors on perd la complétude du prouveur).

Si la liste de clauses est insatisfaisable, on obtient une preuve utilisant les règles de copie, factorisation et résolution binaire, annotée par les unificateurs utilisés.

5.6 Exercices

Exercice 80 (Domaine de Herbrand) *Soient Σ la signature composée de la constante a et des symboles de fonctions f et g respectivement à un et deux arguments.*

— *Donner 5 éléments différents du domaine de Herbrand de cette signature.*
— *Définir inductivement ce domaine.*

□

Exercice 81 (Signature, domaine et base de Herbrand) *Pour chacun de ces ensembles de formules :*

— $\Gamma_1 = \{P(x) \vee Q(x) \vee R(x), \neg P(a), \neg Q(b), \neg R(c)\}$.
— $\Gamma_2 = \{P(x), \neg Q(x), \neg P(f(x)) \vee Q(f(x))\}$.
— $\Gamma_3 = \{P(x), Q(f(x)), \neg R(f(f(x))), \neg P(f(f(x))) \vee \neg Q(x) \vee R(f(x))\}$.

1. *Donner la signature, le domaine de Herbrand ainsi que la base de Herbrand.*
2. *Prouver si leur fermeture universelle a un modèle ou pas.*

□

Exercice 82 (Méthode de Herbrand) *Utiliser la méthode de Herbrand pour démontrer que l'ensemble de formules suivant est insatisfaisable :*

1. $\forall x R(x, f(x))$
2. $\forall x \forall y (\neg S(x,y) \vee R(x,y) \vee R(y,x))$
3. $\forall x \forall y (S(x,y) \vee \neg R(x,y))$
4. $\forall x \forall y (S(x,y) \vee \neg R(y,x))$
5. $\forall y \neg S(y,a)$

□

Exercice 83 (Méthode de Herbrand,*) *Utiliser la méthode de Herbrand pour démontrer que l'ensemble de formules suivant est insatisfaisable :*

1. $\forall x (Q(x) \vee \neg P(f(x)))$
2. $\forall y (Q(y) \Rightarrow R(y))$
3. $\forall z (\neg P(z) \Rightarrow Q(z) \vee R(z))$
4. $\forall u \neg R(u)$

En particulier, vous devez transformer votre ensemble d'instances insatisfaisable en un ensemble de clauses équivalent. Puis, vous devez démontrer la contradiction via une preuve par résolution propositionnelle à partir de ce dernier ensemble. □

Exercice 84 (Méthode de Herbrand,*) *Soit Γ l'ensemble de formules suivant :*

1. $\neg S(x,y) \vee \neg M(z,x) \vee M(z,y)$
2. $S(x,y) \vee M(f(x,y),x)$
3. $S(x,y) \vee \neg M(f(x,y),y)$
4. $S(c,a)$
5. $S(a,b)$
6. $\neg S(c,b)$

Déterminer un ensemble fini insatisfaisable d'instances fermées de ces formules.
On peut en déduire quelque chose à propos d'un ensemble de formules du 1^{er}ordre : lequel et que peut-on conclure ? □

Exercice 85 (Modèle de Herbrand,*)** *Soit Δ l'ensemble de formules suivant :*

1. $x < y \wedge y < z \Rightarrow x < z$.
2. $\neg(x < x)$.
3. $x < y \Rightarrow x < f(x,y) \wedge f(x,y) < y$.
4. $a < b$.

— *Indiquer un modèle de $\forall(\Delta)$.*

— $\forall(\Delta)$ *a-t-il un modèle sur le domaine de Herbrand engendré par* a, b, f *?*

□

Exercice 86 (Skolémisation et forme clausale) *Skolémiser les formules suivantes (attention aux négations!) puis les mettre en forme clausale.*

1. $\neg(\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x)))$.
2. $\neg(\forall x \forall y \forall z(e(x,y) \wedge e(y,z) \Rightarrow \neg e(x,z)) \Rightarrow \neg \exists x \forall y\ e(x,y))$.
3. $\neg(\neg \forall x P(x) \vee \neg \forall x Q(x) \Rightarrow \neg(\forall x P(x) \wedge \forall x Q(x)))$.
4. $\forall x((\exists y P(x,y) \Rightarrow \exists x Q(x)) \wedge \exists y P(x,y) \wedge \neg \exists x Q(x))$.
5. $\neg(\exists x \forall y \forall z((P(y) \Rightarrow Q(z)) \Rightarrow (P(x) \Rightarrow Q(x))))$.

□

Exercice 87 (Unification) *Les termes suivants sont-ils unifiables ? Si la réponse est oui, donner leur unificateur le plus général, sinon justifier la réponse négative.*

— $h(g(x), f(a,y), z)$ *et* $h(y, z, f(u,x))$.
— $h(g(x), f(a,y), z)$ *et* $h(y, z, f(u, g(x)))$.

□

Exercice 88 (Unification) *Donner les unificateurs les plus généraux des termes suivants s'ils existent :*

1. $R(a, f(x)) = R(y, f(g(y,b)))$,
2. $R(y, f(x)) = R(x, f(g(y,b)))$,
3. $Q(y, f(a), f(x)) = Q(g(a,b), x, f(y))$, *et*
4. $Q(y, f(x), g(y,x)) = Q(x, f(y), g(f(a), y))$.

□

Exercice 89 (Unification) *Donner les unificateurs les plus généraux des termes suivants s'ils existent :*

1. $pair(a, crypt(z,b))$ *et* $pair(x,y)$.
2. $pair(crypt(x,b), crypt(y,b))$ *et* $pair(crypt(a,b), z)$.
3. $crypt(pair(z,a), x)$ *et* $crypt(pair(y, crypt(x,b)), b)$.
4. $crypt(pair(a,z), x)$ *et* $crypt(pair(y, crypt(x,b)), b)$.
5. $f(x, y, g(a,a))$ *et* $f(g(y,y), z, z)$
6. $f(x, y, a)$ *et* $f(y, g(z,z), x)$

□

Exercice 90 (Unification avec plusieurs solutions) *L'équation* $f(g(y), y) = f(u, z)$ *a deux solutions « les plus générales »(rappel : elles sont donc équivalentes). Indiquer ces deux solutions.*

□

Exercice 91 (Forme clausale et instanciation,*) *Montrer que la formule suivante est valide en mettant* sa négation *sous la forme clausale et en trouvant un ensemble contradictoire d'instances des clauses obtenues :*

$$\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x)).$$

□

Exercice 92 (Forme clausale et preuve par résolution,)** *Considérons les formules suivantes :*

1. $H_1 = \exists x P(x) \Rightarrow \forall x P(x)$.
2. $H_2 = \forall x(P(x) \vee Q(x))$.
3. $C = \exists x \neg Q(x) \Rightarrow \forall x P(x)$.

Nous voulons montrer que C est conséquence de H_1 *et* H_2 *par instanciation et par résolution.*

1. *Mettre en forme clausale l'ensemble des trois formules* H_1, H_2, $\neg C$.
2. *Trouver des instances contradictoires des clauses obtenues et montrer par résolution propositionnelle que ces instances sont contradictoires.*

3. *Donner une preuve directe de cette contradiction par factorisation, copie et résolution binaire. Il est possible que seule la dernière règle soit utilisée.*

□

Exercice 93 (Preuve par résolution,)**

— *En utilisant une preuve par factorisation, copie et résolvant binaire prouver que la fermeture universelle de l'ensemble de clauses suivant est insatisfaisable :*

1. $P(f(x)) \vee \neg Q(y,a)$.
2. $Q(a,a) \vee R(x,x,b) \vee S(a,b)$.
3. $S(a,z) \vee \neg R(x,x,b)$.
4. $\neg P(f(c)) \vee R(x,a,b)$.
5. $\neg S(y,z) \vee \neg S(a,b)$.

— *En utilisant une preuve par factorisation, copie et résolvant binaire prouver que la fermeture universelle de l'ensemble de clauses suivant est insatisfaisable :*

1. $P(x)$.
2. $\neg P(y) \vee Q(y,x)$.
3. $\neg Q(x,a) \vee \neg Q(b,y) \vee \neg Q(b,a) \vee \neg P(f(y))$.

— *En utilisant une preuve par factorisation, copie et résolvant binaire prouver que la fermeture universelle de l'ensemble de clauses suivant est insatisfaisable :*

1. $R(x,a)$.
2. $R(y,b)$.
3. $R(z,a+b)$.
4. $\neg(b < x+b) \vee (a+b < a)$.
5. $b < z$.
6. $\neg(a+b < x) \vee \neg R(y,b)$.

Rappel : *le symbole $+$ est plus prioritaire que le symbole $<$, qui est lui-même plus prioritaire que les connecteurs logiques binaires.*

□

Exercice 94 (Unification, résolution) *Soient $\Gamma_1 = \{P(x,f(x,b),u), \neg P(g(a),z,h(z))\}$ l'ensemble des* deux *clauses unitaires et Γ_2 l'ensemble des* deux *clauses unitaires $\{P(x,f(x,b),u), \neg P(g(z),z,h(z))\}$. Les ensembles $\forall(\Gamma_1)$ et $\forall(\Gamma_2)$ sont-ils satisfaisables ou insatisfaisables ?* □

Exercice 95 (Skolémisation et preuve par résolution) *Le but de cet exercice est de démontrer le syllogisme suivant :*

$$\forall x(homme(x) \Rightarrow mortel(x)) \wedge homme(Socrate) \Rightarrow mortel(Socrate)$$

— *Skolémiser la* **négation** *du syllogisme.*
— *Transformer la forme de Skolem obtenue en forme clausale.*
— *Démontrer par instanciation que la* **négation** *du syllogisme est une contradiction.*
— *Démontrer en utilisant une preuve par factorisation, copie et résolution binaire que la négation du syllogisme est une contradiction.*

□

Exercice 96 (Forme clausale et preuve par résolution,)** *Considérons les formules suivantes :*

1. $A_1 = \exists u \forall v(P(u) \wedge (R(v) \Rightarrow Q(u,v)))$.
2. $A_2 = \forall u \forall v(\neg P(u) \vee \neg S(v) \vee \neg Q(u,v))$.
3. $A_3 = \exists v(R(v) \wedge S(v))$.

Montrer que la liste de ces trois formules est contradictoire par résolution :

1. *Mettre en forme clausale les trois formules A_1, A_2, A_3 .*

2. *Trouver des instances contradictoires des clauses obtenues et montrer cette contradiction par résolution propositionnelle.*
3. *Faire une preuve directe de $\perp$ par factorisation, copie et résolution binaire.*

□

Exercice 97 (Cohérence de la résolution au premier ordre) *Prouver la cohérence de la résolution au premier ordre, c'est-à-dire :*

Soit Γ un ensemble de clauses et C une clause. Si $\Gamma \vdash_{1fcb} C$ alors $\forall(\Gamma) \models \forall(C)$.

□

Exercice 98 (Forme clausale et preuve par résolution,)** *Considérons les formules suivantes :*

1. $E_1 = \forall x(x = x)$.
2. $E_2 = \forall x \forall y(x = y \Rightarrow y = x)$.
3. $E_3 = \forall x \forall y \forall z \forall t(x = y \wedge z = t \Rightarrow (x \in z \Rightarrow y \in t))$.
4. $C = \forall y \forall z(y = z \Rightarrow \forall x(x \in y \Leftrightarrow x \in z))$.

E_1, E_2, E_3 sont des axiomes de l'égalité : E_1 exprime que l'égalité est réflexive, E_2 qu'elle est symétrique, E_3 que c'est une congruence relativement à l'appartenance. La transitivité de l'égalité ne sert pas dans cet exercice. C exprime que deux ensembles égaux ont les mêmes éléments. Nous notons que dans cet exercice, le symbole égal est traité comme un symbole de prédicat « ordinaire » dont le sens n'est pas fixé comme étant l'identité, mais que nous « définissons » par ses propriétés. Autrement dit l'interdiction de l'égalité dans les formules, c'est en réalité l'interdiction de l'égalité comme symbole de sens fixé. Montrer que C est conséquence des axiomes de l'égalité.

1. *Mettre en forme clausale les trois formules E_1, E_2, E_3, $\neg C$.*
2. *Trouver des instances contradictoires des clauses obtenues.*
3. *Faire une preuve directe de $\perp$ par factorisation, copie et résolution binaire.*

□

Exercice 99 (Forme clausale et preuve par résolution,)** *Considérons les formules suivantes :*

1. $H_1 = \forall u(\exists v R(u,v) \Rightarrow R(u, f(u)))$.
2. $H_2 = \forall u \exists v R(u,v)$.
3. $H_3 = \exists u R(f(f(u)), u)$.
4. $C = \exists u \exists v \exists w(R(u,v) \wedge R(v,w) \wedge R(w,u))$.

Montrer que C est conséquence de H_1, H_2, H_3.

1. *Mettre en forme clausale les trois formules H_1, H_2, H_3, $\neg C$.*
2. *Trouver des instances contradictoires des clauses obtenues (montrez la contradiction par résolution propositionnelle).*
3. *Faire une preuve directe de cette contradiction par factorisation, copie et résolution binaire.*

□

Exercice 100 (Forme clausale et preuve par résolution,)** *Soit $P(x,y)$ une formalisation de « x est père de y ». Soit $A(x,y)$ une formalisation de « x est aïeul de y ». Soient les trois formules :*

1. $H_1 = \forall x \exists y P(x,y)$.
2. $H_2 = \forall x \forall y \forall z(P(x,y) \wedge P(y,z) \Rightarrow A(x,z))$.
3. $C = \forall x \exists y A(x,y)$.

Montrons que C est conséquence de H_1, H_2, de la façon suivante, transformer $H_1, H_2, \neg C$ en leur forme clausale, puis trouver des instances contradictoires des clauses obtenues (montrez la contradiction par résolution propositionnelle). Enfin, dériver la clause vide par factorisation, copie, résolution binaire. □

Exercice 101 (Formule de Bernays-Schonfinkel,*)** *Une formule BS (de Bernays-Schonfinkel) est une formule fermée,* sans *symbole de fonction, de la forme $\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_p$ B où B est sans quantificateur. Expliquer pourquoi nous pouvons décider la satisfaisabilité d'une telle formule.*

□

Chapitre 6

Déduction naturelle au premier ordre : quantificateurs, copie et égalité

Sommaire

6.1 Règles pour la logique du premier ordre **115**
6.1.1 Règles des quantificateurs 116
6.1.2 Copie 119
6.1.3 Les règles de l'égalité 119
6.2 Tactiques de preuves **119**
6.2.1 Raisonner en avant avec une hypothèse d'existence 120
6.2.2 Raisonner en arrière pour généraliser 120
6.2.3 Un exemple d'application des tactiques 120
6.3 Cohérence du système **122**
6.4 Exercices **124**

En plus des règles du chapitre 3, nous ajoutons les règles concernant les quantificateurs, la copie et l'égalité. Notre système comporte toujours une seule règle pour enlever les hypothèses (la règle d'introduction de l'implication). Les définitions de brouillon de preuve, environnement, contexte, formule utilisable restent inchangées. Nous montrons la cohérence des règles de notre système, mais nous admettons sans preuve, que ce système est complet : des preuves de complétude pour des systèmes de règles similaires peuvent être trouvées dans les livres [2, 11]. Contrairement au cas propositionnel, *il n'y a aucun algorithme* pour décider si une formule est valide ou non valide (preuve donnée par Alonzo Church et Alan Türing en 1936 et 1937 [4, 24]). Autrement dit, en admettant l'équivalence entre prouvable (sans environnement) et valide, il n'y a pas d'algorithme qui, étant donné une formule, puisse nous en construire la preuve, ou nous avertir que cette formule n'a pas de preuve.

Plan : Nous étendons d'abord les règles de la déduction naturelle proposées pour la logique propositionnelle. Nous présentons ensuite deux nouvelles tactiques pour aider à la rédaction de preuves. Enfin, nous prouvons la cohérence de notre système.

6.1 Règles pour la logique du premier ordre

Notre système de règles se divise en plusieurs familles de règles :
— les règles « propositionnelles »,
— les règles d'introduction et d'élimination de quantificateur,
— les règles régissant l'égalité,
— et une règle de copie.

L'ensemble des règles définies dans la section 3 page 57 peuvent être utilisées dans notre système de déduction. Nous renvoyons le lecteur à la table 3.1 page 59, qui récapitule l'ensemble de ces règles. Dans la suite de cette section, nous présentons uniquement les règles de déduction naturelle dédiées à la logique du premier ordre.

6.1.1 Règles des quantificateurs

L'ensemble des règles d'introduction et d'élimination de quantificateur est présenté dans la table 6.1. Dans cette table, A et B sont des formules, x est une variable, t est un terme. Notez que contrairement aux règles « propositionnelles », l'usage de ces nouvelles règles est contraint par des conditions d'emploi, qui utilisent les notions de *variable libre* (Définition 4.2.3 page 77) et de *terme libre pour une variable* (Définition 4.3.34 page 82).

Dans la suite, nous détaillons ces règles et nous illustrons leur usage sur des exemples et des contre-exemples montrant les erreurs occasionnées par le non respect des conditions d'emploi.

Règles	**Conditions d'emploi**
$\dfrac{\forall x A}{A<x:=t>}\ \forall E$	
$\dfrac{A}{\forall x A}\ \forall I$	
$\dfrac{\exists x A \quad (A \Rightarrow B)}{B}\ \exists E$	
$\dfrac{A<x:=t>}{\exists x A}\ \exists I$	

TABLE 6.1 – Règles d'introduction et d'élimination des quantificateurs.

6.1.1.1 Règles pour le quantificateur universel

La règle d'élimination du quantificateur universel, notée $\forall E$, signifie que si la formule A est vraie pour toute valeur de x alors toute instance de A où x est remplacé par **un terme** t **libre pour** x est vraie. Notez que cette règle est une simple application du corollaire 4.3.38, page 83.

$$\frac{\forall x A}{A<x:=t>}\ \forall E$$

Nous justifions ci-dessous les conditions d'emploi de cette règle par un exemple montrant qu'un usage incorrect de la règle peut conduire à prouver une formule non-valide.

Exemple 6.1.1 *Nous montrons qu'un usage incorrect de la* ***règle*** $\forall E$ *conduit à « prouver » une formule non valide.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $\forall x \exists y P(x,y)$	
1	*2*	$\exists y P(y,y)$	$\forall E$ *1, y ERREUR*
	3	`Donc` $\forall x \exists y P(x,y) \Rightarrow \exists y P(y,y)$	$\Rightarrow I$ *1,2*

À la ligne 2, nous n'avons pas respecté les conditions d'applications de la règle $\forall E$ *car le terme y n'est pas libre pour x dans la formule* $\exists y P(x,y)$*. Nous donnons une interprétation qui est un contre-modèle de la « conclusion » : soit I l'interprétation de domaine* $\{0,1\}$ *avec* $P_I =$

La règle d'introduction du quantificateur universel, notée $\forall I$, signifie que si nous avons pu déduire la formule A indépendemment de la valeur de x, nous pouvons généraliser en déduisant $\forall x A$. Ainsi, il est nécessaire que x **ne soit libre ni dans l'environnement de la preuve, ni dans le contexte de la ligne où nous avons déduit** A.

$$\frac{A}{\forall x A}\ \forall I$$

Nous illustrons maintenant l'emploi de la règle d'introduction du quantificateur universel avec un exemple d'usage correct (exemple 6.1.2) et un exemple d'usage incorrect (exemple 6.1.3) de la règle.

Exemple 6.1.2 *Nous prouvons* $\forall y P(y) \wedge \forall y Q(y) \Rightarrow \forall x(P(x) \wedge Q(x))$.

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $\forall y P(y) \wedge \forall y Q(y)$	
1	*2*		
1	*3*		
1	*4*		
1	*5*		
1	*6*		
1	*7*		
	8	`Donc` $\forall x P(x) \wedge \forall x Q(x) \Rightarrow \forall x(P(x) \wedge Q(x))$	$\Rightarrow$ *I 1,7*

Exemple 6.1.3 *Nous montrons qu'un usage incorrect de la* ***règle*** $\forall I$ *conduit à « prouver » une formule non valide.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $P(x)$	
1	*2*	$\forall x P(x)$	$\forall I$ *1 ERREUR*
	3	`Donc` $P(x) \Rightarrow \forall x P(x)$	$\Rightarrow$ *I 1,2*

À la ligne 2, nous n'avons pas respecté les conditions d'applications de la règle $\forall I$ *car la prémisse* $P(x)$ *est établie dans le contexte* $P(x)$*, ce qui interdit de généraliser sur x. Nous donnons un contre-modèle de la « conclusion » :*

6.1.1.2 Règles pour le quantificateur existentiel

La règle d'élimination du quantificateur existentiel, notée $\exists E$, signifie que si $\exists x A$ est vraie et que nous pouvons déduire $A \Rightarrow B$ indépendemment de la valeur de x, alors nous pouvons déduire B, si la formule B ne dépend, elle non plus, de la valeur de x. Ainsi, il est nécessaire que x **ne soit libre ni dans l'environnement de la preuve, ni dans** B**, et ni dans le contexte de la ligne** $A \Rightarrow B$**.**

$$\frac{\exists x A \quad (A \Rightarrow B)}{B} \ \exists E$$

La condition d'emploi de cette règle étant complexe, nous illustrons l'emploi de la règle avec deux exemples d'usage incorrect.

Exemple 6.1.4 *Nous montrons qu'un usage incorrect de la* ***règle*** $\exists E$ *conduit à « prouver » une formule non valide.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $\exists x P(x) \wedge (P(x) \Rightarrow \forall y Q(y))$	
1	*2*	$\exists x P(x)$	$\wedge E1$ *1*
1	*3*	$P(x) \Rightarrow \forall y Q(y)$	$\wedge E2$ *1*
1	*4*	$\forall y Q(y)$	$\exists E$ *2,3 ERREUR*
	5	`Donc` $\exists x P(x) \wedge (P(x) \Rightarrow \forall y Q(y)) \Rightarrow \forall y Q(y)$	$\Rightarrow$ *I 1,4*

Nous n'avons pas respecté la condition que le contexte de la prémisse $P(x) \Rightarrow \forall y Q(y)$ ne doit pas dépendre de x. Il est clair que la conclusion obtenue n'est pas valide. Nous donnons une assignation (I, e) qui est un contre-modèle de cette « conclusion » :

Exemple 6.1.5 *Nous montrons qu'un usage incorrect de la* ***règle*** *$\exists E$ conduit à « prouver » une formule non valide.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $\exists x P(x)$	
1,2	*2*	`Supposons` $P(x)$	
1	*3*	`Donc` $P(x) \Rightarrow P(x)$	$\Rightarrow$ *I 2,2*
1	*4*	$P(x)$	$\exists$*E 1,3 ERREUR*
1	*5*	$\forall x P(x)$	$\forall$*I 4*
	6	`Donc` $\exists x P(x) \Rightarrow \forall x\, P(x)$	$\Rightarrow$ *I 1,5*

La conclusion de la règle $\exists E$ est $P(x)$, contrairement à la condition d'application de cette règle qui impose que la conclusion ne doit pas dépendre de x. Nous donnons une interprétation qui est un contre-modèle de la cette « conclusion » :

La règle d'introduction du quantificateur existentiel, notée $\exists I$, signifie que si une instance de la formule A où x est remplacé par **un terme** t **libre pour** x est vraie, alors la formule $\exists x A$ est vraie. Notez que cette règle est une simple application du corollaire 4.3.38, page 83. Ce qui justifie la condition d'emploi. Pour s'en convaincre, il suffit d'étudier l'exemple 4.3.37, qui précède le corollaire 4.3.38.

Ci-dessous, nous donnons un exemple d'usage correct de la règle d'introduction du quantificateur existentiel en démontrant l'une des lois de De Morgan.

Exemple 6.1.6 (Lois de De Morgan) *Nous montrons que pour toute formule A, $\neg\forall x A \Rightarrow \exists x \neg A$.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $\neg\forall x A$	
1,	*2*		
1,	*3*		
1,	*4*		
1,	*5*		
1,	*6*		
1,	*7*		
1,	*8*		
1,	*9*		
1	*10*		
1	*11*		
	12	`Donc` $\neg\forall x A \Rightarrow \exists x \neg A$	$\Rightarrow$ *I 1,11*

6.1.2 Copie

La règle de *copie* consiste à déduire d'une formule A, une autre formule A' égale au changement près des variables liées, au sens de la définition 4.4.5, page 88. Par exemple, $\forall x \exists y P(x,y)$ est une copie de $\forall y \exists x P(y,x)$.

$$\frac{A'}{A}\ \text{Copie}$$

Notez qu'il n'y pas de condition d'emploi pour cette règle.

6.1.3 Les règles de l'égalité

Deux règles caractérisent l'égalité : un terme est égal à lui-même (règle de la *réflexivité*) et si deux termes sont égaux, nous pouvons les remplacer l'un par l'autre (règle de la *congruence*).

$\dfrac{}{t=t}\ Réflexivité$	
$\dfrac{s=t \quad A<x:=s>}{A<x:=t>}\ Congruence$	

Nous remarquons que la première règle n'a pas de prémisse. C'est ce que nous appelons aussi un *axiome*. Nous remarquons aussi que les conditions d'emploi de la deuxième règle sont similaires à celles des règles $\forall E$ et $\exists I$. Ces conditions d'emploi se justifient de la même manière que précédemment.

Nous donnons maintenant deux exemples d'application des règles d'égalités.

Exemple 6.1.7 *Prouvons que $s=t \Rightarrow t=s$, autrement dit prouvons que l'égalité est symétrique.*

contexte	*numéro*	*ligne*	*règle*
1	*1*	`Supposons` $s=t$	
1	*2*		
1	*3*		
	4	`Donc` $s=t \Rightarrow t=s$	

Exemple 6.1.8 *Prouvons que $s=t \wedge t=u \Rightarrow s=u$, autrement dit prouvons que l'égalité est transitive.*

1	*1*	`Supposons` $s=t \wedge t=u$	
1	*2*		
1	*3*		
1	*4*		
	5	`Donc` $s=t \wedge t=u \Rightarrow s=u$	

6.2 Tactiques de preuves

Nous instroduisons deux nouvelles tactiques pour l'application des règles $\forall I$ et $\exists E$ et nous illustrons ces tactiques avec un exemple.

6.2.1 Raisonner en avant avec une hypothèse d'existence

Soient Γ un ensemble de formules, x une variable, A et C des formules. Supposons que nous cherchons une preuve de C dans l'environnement $\Gamma, \exists x A$.

1. Supposons que x n'est libre ni dans Γ, ni dans C. Dans ce cas, la preuve peut toujours s'écrire :

`Supposons` A	
preuve de C dans l'environnement Γ, A	
`Donc` $A \Rightarrow C$	$\Rightarrow I$ 1,_
C	$\exists E$

2. Supposons que x soit libre dans Γ ou C. Nous choisissons une variable y « nouvelle », c'est-à-dire non libre dans Γ, C et absente de A, puis nous nous ramenons au cas précédent, via la règle de copie. La preuve s'écrit alors :

$\exists y A{<}x := y{>}$	Copie de $\exists x A$
`Supposons` $A{<}x := y{>}$	
preuve de C dans l'environnement $\Gamma, A{<}x := y{>}$	
`Donc` $A{<}x := y{>} \Rightarrow C$	$\Rightarrow I$ 1,_
C	$\exists E$

La recherche de la preuve initiale a été réduite à la recherche d'une preuve dans un environnement plus simple. C'est exactement le mode de raisonnement appliqué dans les cours de mathématiques quand nous cherchons une preuve d'une formule C avec l'hypothèse $\exists x P(x)$. Nous introduisons une constante « nouvelle » a vérifiant $P(a)$ et nous prouvons C sous l'hypothèse $P(a)$.

6.2.2 Raisonner en arrière pour généraliser

Nous reprenons les notations du paragraphe précédent. Supposons que nous cherchons une preuve de $\forall x A$ dans l'environnement Γ.

1. Supposons que x n'est pas libre dans Γ. Dans ce cas, la preuve peut toujours s'écrire :

preuve de A dans l'environnement Γ	
$\forall x A$	$\forall I$

2. Supposons que x est libre dans Γ. Nous choisissons une variable y « nouvelle », c'est-à-dire non libre dans Γ, puis nous nous ramenons au cas précédent, via la règle de copie. La preuve s'écrit alors :

preuve de $A{<}x := y{>}$ dans l'environnement Γ	
$\forall y A{<}x := y{>}$	$\forall I$
$\forall x A$	Copie de la formule précédente

La recherche de la preuve initiale a été réduite à la recherche d'une preuve d'une formule plus simple dans le même environnement. C'est exactement le mode de raisonnement appliqué dans les cours de mathématiques quand nous cherchons une preuve de $\forall x P(x)$. Nous introduisons une constante « nouvelle » a et nous prouvons $P(a)$. Puis nous ajoutons : puisque le choix de a est arbitraire, nous avons $\forall x P(x)$.

6.2.3 Un exemple d'application des tactiques

Nous notons « il existe un et un seul x » par $\exists! x$. Formellement, $\exists! x P(x)$ signifie $\exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$. En séparant l'existence de x et son unicité, nous pouvons aussi définir $\exists! x P(x)$ par $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y)$. Ces deux définitions sont bien sûr équivalentes et nous montrons formellement que la première implique la deuxième. Comme la preuve est longue, il faut apprendre à décomposer les preuves.

Plan de la preuve Nous appliquons les deux tactiques suivantes :
- Pour prouver $A \Rightarrow B$, supposer A et déduire B
- Pour prouver $A \wedge B$, prouver A et prouver B.

1	`Supposons` $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$	
	preuve de $\exists x P(x)$ dans l'environnement de (1)	
	preuve de $\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ dans l'environnement de (1)	
	$\exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$	$\wedge I$
	`Donc` $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y)) \Rightarrow \exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$	$\Rightarrow I$

Application de la tactique utilisant une hypothèse d'existence

Nous cherchons une preuve de $\exists x P(x)$ dans l'environnement de $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$. Nous appliquons la tactique « raisonner en avant en présence d'une hypothèse existentielle ».

référence	formule	
i	$\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$	
1	`Supposons` $P(x) \wedge \forall y(P(y) \Rightarrow x = y)$	
2	$P(x)$	$\wedge E1$ 1
3	$\exists x P(x)$	$\exists I$ 2, x
4	`Donc` $P(x) \wedge \forall y(P(y) \Rightarrow x = y) \Rightarrow \exists x P(x)$	$\Rightarrow I$ 1,2
5	$\exists x P(x)$	$\exists E$ i, 3

Application de la tactique pour obtenir une conclusion générale

Nous cherchons une preuve de $\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$ dans l'environnement de $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$. Nous appliquons, dans l'ordre ci-dessous, les tactiques suivantes :
- « raisonner en avant en utilisant d'une hypothèse existentielle ».
- Pour prouver $A \Rightarrow B$, supposer A et déduire B
- « raisonner en arrière pour obtenir une conclusion générale ».

environnement			
référence		formule	
i		$\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$	
contexte	numéro	preuve	règle
1	1	`Supposons` $P(x) \wedge \forall y(P(y) \Rightarrow x = y)$	
1	2		
1	3		
1	4		
1	5		
1	6		
1	7		
1	8		
1	9		
1	10		
1	11		
1	12		
1	13		
1	14		
	15		
	16	$\forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y)$	

La tactique « utiliser une hypothèse d'existence » sert à produire la ligne 16. La tactique « introduire une implication » sert à produire la ligne 15 : elle introduit l'hypothèse (1) dans laquelle x est une variable libre. Donc, pour appliquer la tactique « obtenir une conclusion générale », il faut changer de variable dès la ligne 2. Observer aussi que la formule, ligne 3, est instanciée aux lignes 5 et 8.

Comme sur cet exemple, toute la difficulté des preuves est concentrée autour des règles $\forall I$ et $\exists E$:

— dans le raisonnement en avant, il faut trouver les bonnes instanciations des formules commençant par un quantificateur existentiel.

— dans le raisonnement en arrière, il faut trouver la bonne instance permettant de déduire une formule commençant par un quantificateur universel.

6.3 Cohérence du système

Nous commençons par montrer deux propriétés à propos des quantificateurs existensiel et universel. Enfin nous montrons la cohérence de notre système de déduction.

Propriété 6.3.1 *Soient Γ un ensemble de formules, x une variable et A une formule. Supposons que x ne soit pas libre dans Γ, alors nous avons : $\Gamma \models A$ si et seulement si $\Gamma \models \forall x A$.*

Preuve :

□

Si nous observons d'un œil critique cette preuve, nous observons que c'est une paraphrase, dans un autre formalisme, de la loi $\forall I$. C'est l'équivalence ci-dessus, qui explique que la tactique « raisonner en arrière pour généraliser », est une tactique utilisable sans risque. En effet, en admettant la complétude du système, lorsque x n'est pas libre dans Γ, il y a une preuve de A dans l'environnement Γ si et seulement s'il a une preuve de $\forall x A$ dans ce même environnement.

Propriété 6.3.2 *Soient Γ un ensemble de formules, x une variable, A et B deux formules. Supposons que x ne soit libre ni dans Γ, ni dans B, alors nous avons : $\Gamma \models A \Rightarrow B$ si et seulement si $\Gamma \models (\exists x A) \Rightarrow B$*

Preuve :

$\Rightarrow$ Supposons que $\Gamma \models A \Rightarrow B$. Soit (I,e) une assignation modèle de Γ. Puisque x n'est pas libre dans Γ, pour tout état f identique à e sauf pour la valeur de x, (I,f) et (I,e) donnent la même valeur aux formules de Γ, donc (I,f) est modèle de Γ. Puisque $\Gamma \models A \Rightarrow B$, pour tout état f identique à e sauf pour la valeur de x, (I,f) est modèle de $A \Rightarrow B$. Supposons que (I,e) est modèle de $\exists x A$, il existe g identique à e sauf pour la valeur de x tel que (I,g) est modèle de A. Puisque pour tout état f identique à e sauf pour la valeur de x, (I,f) est modèle de $A \Rightarrow B$, alors (I,g) est modèle de B. Puisque x n'est pas libre dans B, (I,e) est modèle de B.

$\Leftarrow$ Supposons que $\Gamma \models (\exists x A) \Rightarrow B$. Puisque la formule $A \Rightarrow (\exists x A)$ est valide (d'après le corollaire 4.3.38 page 83), nous avons $\Gamma \models A \Rightarrow B$.

□

Si nous observons d'un œil critique cette preuve, nous voyons que c'est une paraphrase, dans un autre formalisme, de la loi $\exists E$. C'est l'équivalence ci-dessus, qui explique que la tactique « raisonner en avant avec une hypothèse d'existence », est une tactique utilisable sans risque. En effet, en admettant la complétude du système, lorsque x n'est libre ni dans Γ, ni dans B, il y a une preuve de B dans l'environnement Γ, A si et seulement s'il a une preuve de B dans l'environnement $\Gamma, \exists x A$.

Théorème 6.3.3 (Cohérence de la déduction) *Si une formule est déduite d'un environnement de formules alors elle en est une conséquence.*

Preuve : La preuve obéit au même plan que celle du même théorème dans le cas propositionnel (voir théorème 3.3.1 page 65) et nous en reprenons les notations en traitant uniquement le cas des nouvelles règles.

Soit Γ un ensemble de formules. Soit P une preuve de A dans cet environnement. Soient C_i la conclusion et H_i le contexte de la i-ème ligne de la preuve P. Rappelons que les lignes d'une preuve sont numérotées à partir de 1 et que H_0 est la liste vide.

Notons par Γ, H_i l'ensemble des formules de l'ensemble Γ et de la liste H_i.

Cas de base : Supposons que A est déduite de Γ par la preuve vide. Alors A est élément de Γ, donc $\Gamma \models A$. Puisque H_0 est la liste vide, nous pouvons conclure : $\Gamma, H_0 \models A$.

Induction : Supposons que pour toute ligne $i < k$, $\Gamma, H_i \models C_i$. Montrons que $\Gamma, H_k \models C_k$.

Nous examinons uniquement le cas des nouvelles règles et pour simplifier nous ne faisons pas de distinctions entre deux formules égales aux abréviations près de la négation et de l'équivalence.

— Supposons que $C_k = \forall xA$ et que cette ligne ait été déduite, par la règle $\forall I$, de la formule A avec $A = C_i$ et $0 \leq i < k$ ou $A \in \Gamma$. Si $A = C_i$ et $0 < i < k$, par hypothèse de récurrence nous avons, $\Gamma, H_i \models A$. Si $A \in \Gamma$ alors $\Gamma \models A$. Puisque H_0 est la liste vide, il existe i où $0 \leq i < k$ tel que $\Gamma, H_i \models A$. Vu les conditions d'application de la règle, x n'est pas libre dans Γ, H_i. Donc, d'après la propriété 6.3.1 page ci-contre, nous avons aussi $\Gamma, H_i \models \forall xA$. Puisque la ligne i est utilisable sur la ligne $k-1$ et que H_0 est la liste vide, H_i est préfixe de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models C_k$.

— Supposons que $C_k = A{<}x := t{>}$ et que cette ligne ait été déduite, par la règle $\forall E$, de la formule $\forall xA$ avec $\forall xA = C_i$ et $0 < i < k$ ou $\forall xA \in \Gamma$. Par hypothèse de récurrence ou parce que H_0 est la liste vide, il existe i où $0 \leq i < k$ tel que $\Gamma, H_i \models \forall xA$. D'après les conditions d'application de la règle, le terme t est libre pour la variable x dans la formule A. Donc, d'après le corollaire 4.3.38 page 83, la formule $\forall xA \Rightarrow A{<}x := t{>}$ est valide et par suite $\Gamma, H_i \models A{<}x := t{>}$. Puisque la ligne i est utilisable sur la ligne $k-1$, et que H_0 est la liste vide, H_i est préfixe de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models C_k$.

— Supposons que $C_k = \exists xA$ et que cette ligne ait été déduite, par la règle $\exists I$, de la formule $A{<}x := t{>}$ avec $A{<}x := t{>} = C_i$ et $0 < i < k$ ou $A{<}x := t{>} \in \Gamma$. Par hypothèse de récurrence ou parce H_0 est la liste vide, il existe i où $0 \leq i < k$ tel que $\Gamma, H_i \models A{<}x := t{>}$. D'après les conditions d'applications de la règle, le terme t est libre pour la variable x dans la formule A. Donc, d'après le corollaire 4.3.38 page 83, la formule $A{<}x := t{>} \Rightarrow \exists xA$ est valide et par suite $\Gamma, H_i \models \exists xA$. Puisque la ligne i est utilisable sur la ligne $k-1$, et que H_0 est la liste vide, H_i est préfixe de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models C_k$.

— Supposons que $C_k = B$ et que cette formule ait été déduite, par la règle $\exists E$, de la formule $\exists xA$ avec $\exists xA = C_i$ et $0 < i < k$ ou $\exists xA \in \Gamma$ et de la formule $A \Rightarrow B$ avec $A \Rightarrow B = C_j$ et $0 < j < k$ ou $A \Rightarrow B \in \Gamma$. Par hypothèse de récurrence ou parce H_0 est la liste vide, il existe i et j tels que $0 \leq i < k$, $0 \leq j < k$, $\Gamma, H_i \models \exists xA$ et $\Gamma, H_j \models A \Rightarrow B$. Vu les conditions d'application de la règle, x n'est libre ni dans Γ, H_j, ni dans B. Donc, d'après la propriété 6.3.2 page précédente, nous avons aussi $\Gamma, H_j \models (\exists xA) \Rightarrow B$. Puisque les lignes i et j sont utilisables sur la ligne $k-1$, et que H_0 est la liste vide, H_i et H_j sont préfixes de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models \exists xA$ et $\Gamma, H_k \models (\exists xA) \Rightarrow B$. Par suite $\Gamma, H_k \models C_k$.

— Supposons que $C_k = A'$ et que cette formule ait été déduite, par la règle de copie de la formule A avec $A = C_i$ et $0 < i < k$ ou $A \in \Gamma$. Par hypothèse de récurrence ou parce H_0 est la liste vide, il existe i tel que $0 \leq i < k$, $\Gamma, H_i \models A$. Puisque, d'après le théorème 4.4.6 page 88, les formules A et A' sont équivalentes, nous avons $\Gamma, H_i \models A'$. Puisque la ligne i est utilisable sur la ligne $k-1$, et que H_0 est la liste vide, H_i est préfixe de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models C_k$.

— Supposons que $C_k = (t = t)$. Puisque cette formule est valide (dans le sens attribué à l'égalité), $\Gamma, H_k \models C_k$.

— Supposons que $C_k = A{<}x := t{>}$ et que cette ligne ait été déduite, par la règle de congruence, de la formule $s = t$ avec $(s = t) = C_i$ et $0 < i < k$ ou $(s = t) \in \Gamma$ et de la formule $A{<}x := s{>}$ avec $A{<}x := s{>} = C_j$ et $0 < j < k$ ou $A{<}x := s{>} \in \Gamma$. Par hypothèse de récurrence ou parce H_0 est la liste vide, il existe i et j tels que $0 \leq i < k$, $0 \leq j < k$, $\Gamma, H_i \models (s = t)$ et $\Gamma, H_j \models A{<}x := s{>}$. Puisque les lignes i et j sont utilisables sur la ligne $k-1$, et que H_0 est la liste vide, H_i et H_j sont préfixes de H_{k-1}. Puisque le contexte n'est pas modifié par la ligne k, nous avons $H_{k-1} = H_k$, donc $\Gamma, H_k \models (s = t)$ et $\Gamma, H_k \models A{<}x := s{>}$. D'après le théorème 4.3.36 page 83 et les conditions d'application de la règle, nous avons : $s = t, A{<}x := s{>} \models A{<}x := t{>}$. Par suite $\Gamma, H_k \models C_k$.

Puisque la dernère ligne d'une preuve a un contexte vide, nous obtenons que sa conclusion est conséquence de Γ. □

6.4 Exercices

Exercice 102 (Déductions naturelles) *Prouver en déduction naturelle du premier ordre les formules suivantes :*

1. *le fameux syllogisme, « Tout homme est mortel, Socrate est un homme, donc est mortel », que nous formalisons par* $\forall x(H(x) \Rightarrow M(x)) \wedge H(socrate) \Rightarrow M(socrate)$.
2. $\forall x P(x) \Rightarrow \exists y P(y)$.
3. $\forall x P(x) \Rightarrow \exists x P(x)$.
4. $\forall x(P(x) \wedge Q(x)) \Rightarrow \forall x P(x) \wedge \forall x Q(x)$. *Notez que l'exemple* 6.1.2 page 117 *donne la preuve de la réciproque.*
5. $\exists x(P(x) \vee Q(x)) \Rightarrow \exists x P(x) \vee \exists x Q(x)$.
6. $\forall x(P(x) \Rightarrow Q(x)) \wedge \exists x P(x) \Rightarrow \exists x Q(x)$.
7. $\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x))$. *(**)*
8. $\forall x(P(x) \Rightarrow Q(x)) \wedge \forall x(Q(x) \Rightarrow R(x)) \Rightarrow \forall x(P(x) \Rightarrow R(x))$.
9. $\forall x\ (P(x) \Rightarrow Q(x)) \wedge \exists x\ \neg Q(x) \Rightarrow \exists x\ \neg P(x)$.

Dans cet exercice, les formules $P(x)$ *et* $Q(x)$ *peuvent être remplacées par des formules* quelconques. □

Exercice 103 (Déductions naturelles) *Prouver les formules suivantes :*

1. $\forall x \forall y P(x,y) \Rightarrow \forall y \forall x P(x,y)$.
2. $\exists x \exists y P(x,y) \Rightarrow \exists y \exists x P(x,y)$.
3. $\exists x \forall y P(x,y) \Rightarrow \forall y \exists x P(x,y)$.
4. $\forall x(Q(x) \Rightarrow \forall y(R(y) \Rightarrow P(x,y))) \Rightarrow \forall y(R(y) \Rightarrow \forall x(Q(x) \Rightarrow P(x,y)))$. *(*)*

Dans cet exercice, la formule $P(x,y)$ *peut être remplacée par une formule quelconque. Par contre* $Q(x)$ *peut être remplacée seulement par une formule n'ayant pas y comme variable libre, et* $R(y)$ *peut être remplacée par une formule n'ayant pas x comme variable libre : expliquez la raison de ces contraintes.* □

Exercice 104 (Chercher la faute) *Considérons la formule suivante :*

$$\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x\ (P(x) \wedge Q(x)).$$

Parmi les trois preuves suivantes une seule est correcte par déduction naturelle. Identifier la preuve correcte et justifier pourquoi les deux autres ne le sont pas.

1.

contexte	*numéro*	*preuve*	*règle*
1	*1*	`Supposons` $\exists x\ P(x) \wedge \forall x\ Q(x)$	
1	*2*	$\exists x\ P(x)$	$\wedge E1$ *1*
1	*3*	$\forall x\ Q(x)$	$\wedge E2$ *1*
1,4	*4*	`Supposons` $P(x)$	
1,4	*5*	$Q(x)$	$\forall E$ *3, x*
1,4	*6*	$P(x) \wedge Q(x)$	$\wedge I$ *4,5*
1,4	*7*	$\exists x(P(x) \wedge Q(x))$	$\exists I$ *6, x*
1	*8*	`Donc` $P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow I$ *4,7*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists E$ *2,8*
	10	`Donc` $\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow I$ *1,9*

2.

contexte	*numéro*	*preuve*	*règle*
1	*1*	`Supposons` $\exists x\, P(x) \wedge \forall x\, Q(x)$	
1	*2*	$\exists x\, P(x)$	$\wedge E1$ *1*
1	*3*	$\forall x\, Q(x)$	$\wedge E2$ *1*
1,4	*4*	`Supposons` $P(x)$	
1,4	*5*	$Q(x)$	$\forall E$ *3, x*
1,4	*6*	$P(x) \wedge Q(x)$	$\wedge I$ *4,5*
1	*7*	`Donc` $P(x) \Rightarrow P(x) \wedge Q(x)$	$\Rightarrow I$ *4,6*
1	*8*	$P(x) \wedge Q(x)$	$\exists E$ *2,7*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists I$ *8, x*
	10	`Donc` $\exists x\, P(x) \wedge \forall x\, Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow I$ *1,9*

3.

contexte	*numéro*	*preuve*	*règle*
1	*1*	`Supposons` $\exists x\, P(x) \wedge \forall x\, Q(x)$	
1	*2*	$\exists x\, P(x)$	$\wedge E1$ *1*
1	*3*	$\forall x\, Q(x)$	$\wedge E2$ *1*
1,4	*4*	`Supposons` $P(x)$	
1	*5*	`Donc` $P(x) \Rightarrow P(x)$	$\Rightarrow I$ *4,4*
1	*6*	$P(x)$	$\exists E$ *2,5*
1	*7*	$Q(x)$	$\forall E$ *3, x*
1	*8*	$P(x) \wedge Q(x)$	$\wedge I$ *6,7*
1	*9*	$\exists x(P(x) \wedge Q(x))$	$\exists I$ *8, x*
	10	`Donc` $\exists x\, P(x) \wedge \forall x\, Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow I$ *1,9*

□

Exercice 105 (Copie) *Prouver par déduction naturelle que* $\forall x \forall y P(x,y) \Rightarrow \forall x \forall y P(y,x)$ *en utilisant la règle de copie un minimum de fois.* □

Exercice 106 (Déduction naturelle) *Prouver les formules suivantes grâce à la déduction naturelle (notez que Q est une variable propositionnelle) :*

1. $\forall x(Q \wedge P(x)) \Rightarrow Q \wedge \forall x P(x)$.
2. $Q \wedge \forall x P(x) \Rightarrow \forall x(Q \wedge P(x))$.
3. $\forall x(Q \vee P(x)) \Rightarrow Q \vee \forall x P(x)$. *(**)*
4. $Q \vee \forall x P(x) \Rightarrow \forall x(Q \vee P(x))$.
5. $\exists x(Q \wedge P(x)) \Rightarrow Q \wedge \exists x P(x)$.
6. $Q \wedge \exists x P(x) \Rightarrow \exists x(Q \wedge P(x))$.
7. $\exists x(Q \vee P(x)) \Rightarrow Q \vee \exists x P(x)$.
8. $Q \vee \exists x P(x) \Rightarrow \exists x(Q \vee P(x))$. *(*)*

Dans cet exercice, la formule $P(x)$ *peut être remplacée par une formule quelconque. Par contre, Q peut être remplacée seulement par une formule n'ayant pas x comme variable libre.* □

Exercice 107 (Preuve) *Prouver la formule* $\neg\exists x P(x) \Rightarrow \forall x \neg P(x)$. *Vérifier que* $P(x)$ *peut être remplacée par une formule quelconque. Remarque : cette formule est la réciproque de la formule prouvée à l'exemple* *6.1.6 page 118*. □

Exercice 108 (Égalité) *Prouver les formules suivantes :*

1. $R(a,c) \wedge (a = b) \Rightarrow R(b,c)$.
2. $x = y \Rightarrow f(x,z) = f(y,z)$.
3. $\forall x \exists y(x = y)$.

4. $\exists x \forall y\, x = y \Rightarrow \forall x \forall y\, x = y$. (*)

□

Exercice 109 (Égalité,)** *Prouver que la deuxième définition de « il existe un et un seul x » (voir sous-section 6.2.3 page 120) implique la première, autrement dit prouver la formule :* $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$. □

Exercice 110 (Induction et déduction naturelle,*)** *Nous pouvons définir l'addition grâce aux deux formules :*

(a) $\forall n (n + 0 = n)$.

(b) $\forall n \forall p (n + s(p) = s(n + p))$.

Ces deux formules permettent de faire des additions : nous pouvons grâce à elles prouver que $s(0) + s(0) = s(s(0))$. *Mais elles ne permettent pas de prouver les propriétés générales de l'addition, qui nécessitent d'avoir le principe de récurrence.*

1. *Montrer que des hypothèses (a) et (b), nous ne pouvons pas déduire* $\forall n (0 + n = n)$.
2. *Nous donnons un nom à la propriété ci-dessus et nous posons le principe de récurrence sur cette propriété :*
   (c) $\forall n (P(n) \Leftrightarrow 0 + n = n)$.
   (d) $P(0) \wedge \forall n (P(n) \Rightarrow P(s(n))) \Rightarrow \forall n P(n)$.
   Prouver par la déduction naturelle que $(a) \wedge (b) \wedge (c) \wedge (d) \Rightarrow \forall n P(n)$.

□

Exercice 111 (Examen 2009) *Les preuves demandées ci-dessous devront être justifiées.*

1. *Donner une preuve en* déduction naturelle avec les justifications *de la validité de la formule :*
   $(\exists x\, p(x) \Rightarrow \forall x\, q(x)) \Rightarrow \forall x (p(x) \Rightarrow q(x))$.
2. *Donner une preuve en* déduction naturelle avec les justifications *de la validité de la formule :*
   $\exists x\, p(x) \wedge \forall x (p(x) \Rightarrow p(f(x))) \Rightarrow \exists x\, p(f(f(x)))$.
3. *Notons* $f^n(x)$ *le terme obtenu en appliquant n fois f à x.*
   Par exemple $f^0(x) = x, f^1(x) = f(x), f^2(x) = f(f(x))$.
   Soient Γ, Δ *deux ensembles de formules et* A, B *deux formules. On rappelle que* $\Gamma \vdash A$ *est vrai s'il y a une preuve de A dans l'*environnement Γ.
   On donne ci-dessous des propriétés triviales de la relation $\vdash$.
   Monotonie : si $\Gamma \vdash A$ *et* $\Gamma \subset \Delta$ *alors* $\Delta \vdash A$.
   Composition : si $\Gamma \vdash A$ *et* $\Gamma \vdash A \Rightarrow B$ *alors* $\Gamma \vdash B$.

   (a) Donnez une preuve en déduction naturelle avec les justifications *de ce que la propriété suivante est vraie :*
   $\forall x (p(x) \Rightarrow p(f(x))) \vdash \exists x p(f^n(x)) \Rightarrow \exists x\, p(f^{n+1}(x))$

   (b) Déduire de la propriété ci-dessus, de la monotonie et de la composition que pour tout *entier naturel n :*
   $\exists x\, p(x), \forall x (p(x) \Rightarrow p(f(x))) \vdash \exists x\, p(f^n(x))$.

□

Exercice 112 (Examen 2012) *Prouver les formules suivantes par déduction naturelle au premier ordre.*

1. $\neg \forall x P(x) \vee \neg \exists y Q(y) \Rightarrow \neg (\forall x P(x) \wedge \exists y Q(y))$.
2. $\forall x \forall y (P(y) \Rightarrow R(x)) \Rightarrow \exists y P(y) \Rightarrow \forall x R(x)$.
3. $\neg \forall x \neg P(x) \Rightarrow \exists x P(x)$.

□

Exercice 113 (Examen 2013) *Prouver les formules suivantes par déduction naturelle au premier ordre.*

1. $\exists x (Q(x) \Rightarrow P(x)) \wedge \forall x Q(x) \Rightarrow \exists x P(x)$.
2. $\forall x\, \forall y (R(x,y) \Rightarrow \neg R(y,x)) \Rightarrow \forall x \neg R(x,x)$.

□

Exercice 114 (Quelques questions posées en examen) *Démontrer les formules suivantes par déduction naturelle.*

1. $\exists x (P(x) \vee Q(x)) \wedge \forall x \neg Q(x) \Rightarrow \exists x P(x)$.
2. $\forall x (P(x) \Rightarrow Q(x)) \wedge \exists x (P(x) \wedge R(x)) \Rightarrow \exists x (Q(x) \wedge R(x))$.
3. $\exists x \neg (P(x) \vee \neg P(x)) \Rightarrow \forall x P(x)$.

□

Bibliographie

[1] Collin Allen and Michael Hand. *Logic Primer*. MIT, 2001.

[2] Peter B. Andrews. *An introduction to mathematical logic : to truth through proof*. Academic Press, 1986.

[3] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[4] Alonzo Church. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, 1(1) :40–41, 1936.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[6] René Cori and Daniel Lascar. *Logique mathématique Cours et exercices I. Calcul propositionnel, algèbres de Boole, calcul des prédicats*. Masson, 1993.

[7] René Cori and Daniel Lascar. *Logique mathématique Cours et exercices II. Fonctions récursives, théorème de Gödel, théorie des ensembles, théories des modèles*. Masson, 1993.

[8] René David, Karim Nour, and Christophe Raffali. *Introduction à la logique*. Dunod, 2001.

[9] Martin Davis, George W. Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of The ACM*, 5 :394–397, 1962.

[10] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3) :201–215, 1960.

[11] Herbert B. Enderton. *A mathematical Introduction to Logic*. Academic Press, 2001.

[12] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39 :176–210, 1935. 10.1007/BF01201353.

[13] Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39 :405–431, 1935. 10.1007/BF01201363.

[14] Roger Godement. *Cours d'algèbre*. Hermann, 1973.

[15] Gérard P. Huet. unification in typed lambda calculus. In Corrado Böhm, editor, *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975*, volume 37 of *Lecture Notes in Computer Science*, pages 192–212. Springer, 1975.

[16] Gérard P. Huet. Higher order unification 30 years later. In Victor Carreño, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002, Proceedings*, volume 2410 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2002.

[17] Michael Huth and Mark Ryan. *Logic in Computer Science*. Cambridge University Press, 2004.

[18] Jacquemin. *Logique et Mathématiques 109 exercices corrigées*. Masson, 1994.

[19] Stephen C. Kleene. *Logique Mathématique*. Armand Colin, 1971.

[20] Thierry Lucas, Isabelle Berlanger, and Isabelle De Greef. *Initiation à la logique formelle*. De Boeck, 2003.

[21] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4 :258–282, April 1982.

[22] William McCune. Prover9 and mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005–2010.

[23] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1) :23–41, January 1965.

[24] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42 :230–265, 1937.

Index

$C \tilde{+} D$, 39
F_Σ, 79
T_Σ, 79
$\Gamma[L := 1]$, 41
$|A|$, 13, 75
$\bot$, 12
$\top$, 12
n-expansion, 84
$\Gamma[L := 1]$, 41
$v[L := 1]$, 42

algèbre de Boole, 24
algorithme d'unification, 103
algorithme DPLL, 37, 45
arbre, 13
Aristote, 73
arité, 78
assignation, 15, 80
axiome, 119

Baader, Franz, 102
base de Herbrand, 94
BDDC, 30
Boole, George, 24
brouillon de preuve, 60

Church, Alonzo, 93, 115
clause, 22, 37, 101
clause unitaire, 46
complet, 34
complétude pour la réfutation, 41
composition, 40
conclusion, 61
conflit, 50
conjonction, 12
connecteur, 11, 12
conséquence, 17
consistant, 17
constante, 12, 78
contexte, 60
contradiction, 17
contre-modèle, 17
contre-modèle d'un ensemble de formules, 17
copie d'une clause, 106

Davis, Martin, 37
déclaration de symbole, 78
déduction, 37
diagramme de décision binaire, 30
disjonction, 12
domaine, 79
domaine de Herbrand, 94

éléments d'une clause, 38
ensemble de clauses réduites, 42
ensemble de formules insatisfaisable, 17
ensemble de formules satisfaisable, 17
équivalence, 12
état, 80
état, 80
évaluation, 80
expression, 102

facteur, 106
faux, 12
fermeture universelle, 94
fonction booléenne, 29
forme clausale, 101
Forme clausale d'un ensemble de formules, 101
forme de Skolem, 97
forme normale, 22, 34
forme normale conjonctive, 24
forme normale disjonctive, 23
formule, 12, 75, 79
formule à priorité, 14, 76
formule atomique, 75, 79
formule atomique sur une signature, 79
formule décomposable, 12
formule équivalente, 15
formule fermée, 77
formule insatisfaisable, 17
formule propre, 97
formule satisfaisable, 17
formule stricte, 12
formule sur une signature, 79
formule utilisable, 61
formules égales à un changement près de variables liées, 88

Gentzen, Gerhard, 57, 60, 62
graphe d'implication, 50

Herbrand, Jacques, 93
heuristique JW, 50
heuristique MOMS, 50
heuristique UP, 50
Huet, Gérard, 102

implication, 12
instanciation, 82
interprétation, 79, 80
interprétation d'un ensemble de formules, 80
interprétation de Herbrand, 94
inverse du renommage, 106

Kleene, Stephen Cole, 86

ligne de preuve, 60
littéral, 22, 38
littéral complémentaire, 38
littéral isolé, 46
littéral négatif, 101
littéral positif, 101
Logemann, George W., 37
loi de Peirce, 64
Loveland, Donald W., 37

Martelli, Alberto, 102
modèle d'un ensemble de formules, 16
modèle d'une formule, 16
modus ponens, 59, 73
monôme, 22
monotone, 35
monotonie, 40
Montanari, Ugo, 102

négation, 12

occurence libre, 77
occurence liée, 77
ordre de priorité des connecteurs, 14

parenthèse, 12
Peirce, Charles Sanders, 64
portée de liaison, 77
Prawitz, Dag, 66
preuve, 39, 61
preuve d'une formule, 61
problème 3-SAT, 49
problème SAT, 49
problème NP-complet, 49
Putnam, Hilary, 37

Raymond, Pascal, 12
réduction linéaire, 49
règle, 58
remplacement, 21
renommage, 106
résolution, 37
résolution restreinte, 50
résolution unitaire, 46
résolvant, 38, 39
résolvant au premier ordre, 109
résolvant binaire, 107
Robinson, Alan, 37, 102

sens des formules, 81
sens des formules atomiques, 81
signature, 78
signature associée à un ensemble de formules, 79
signature associée à une formule, 79
Skolem, Thoralf Albert, 97
skolémisation, 97
Snyder, Ayne, 102
Socrate, 73
solution de l'équation, 102
solution la plus générale, 103
solveur SAT, 49
solveur SAT complet, 50
solveur SAT incomplet, 50
sous-clause, 38
sous-formule, 13
stratégie complète, 37, 43
structure paresseuse, 51
substitution, 19
substitution à support fini, 19
support d'une substitution, 19
symbole de fonction, 78
symbole de relations, 78

table de vérité, 15
taille d'une formule, 13
taille d'une formule du premier ordre, 75
taille de preuve, 40
tautologie, 16
terme, 75, 78
terme sur une signature, 78
théorie, 100
Théry, Laurent, 66
tiers-exclus, 19
Türing, Alan, 93, 115

unaire, 78
unificateur le plus général, 103
unification, 102

valeur d'une formule, 15
valide, 16
variable, 12, 78
variable libre, 77
variable liée, 77
variable propositionnelle, 78
vrai, 12

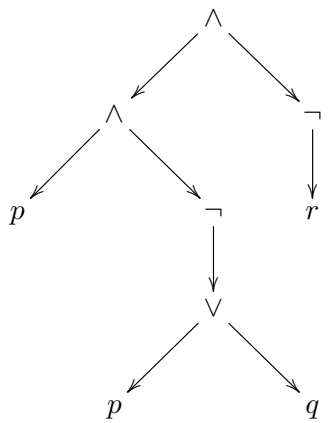
Exercises for Chapter I.1

Exercise 1 (Strict Formulas, Prioritized Formulas)
Which of the following expressions are strict formulas?

1. x
2. $(x$
3. (x)
4. $(x \Rightarrow (y \wedge z)$
5. $(x \Rightarrow (y \wedge z))$
6. $\neg(x \vee y)$
7. $(\neg(x \vee y))$
8. $\neg(x \Rightarrow y) \vee \neg(y \wedge z)$
9. $\neg(x)$
10. $(\neg((x \vee y) \wedge z) \Leftrightarrow (u \Rightarrow v)))$

Exercise 2 (Formulas and Priority)
Prove that every (strict) formula is a prioritized formula.

Exercise 3 (Formulas and Priority)
Consider the following tree representation of the formula $((p \wedge \neg(p \vee q)) \wedge \neg r)$.



Using priority, this formula can be written with fewer parentheses: $p \wedge \neg(p \vee q) \wedge \neg r$ using logical notation, $p\ \overline{p+q}\ \overline{r}$ using boolean notation. Similarly, give the tree representation, logical notation and boolean notation, both using as few parentheses as possible of the following formulas:

1. $(\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b))$.
2. $((\neg a \vee b) \wedge (\neg b \vee a))$.
3. $(((a \wedge b) \wedge c) \vee ((\neg a \wedge \neg b) \wedge \neg c))$.
4. $(p \Rightarrow (q \Rightarrow r))$.
5. $((p \Rightarrow q) \Rightarrow r)$.

Exercise 4 (Formulas and Priority)
Give the tree representation of the following prioritized formulas:

1. $p \Leftrightarrow \neg q \vee r$.
2. $p \vee q \Rightarrow r \wedge s$.
3. $p \vee q \Rightarrow r \Leftrightarrow s$.
4. $p \vee q \wedge r \Rightarrow \neg s$.
5. $p \Rightarrow r \wedge s \Rightarrow t$.
6. $p \vee q \wedge s \vee t$.
7. $p \wedge q \Leftrightarrow \neg r \vee s$.
8. $\neg p \wedge q \vee r \Rightarrow s \Leftrightarrow t$.

Exercise 5 (Height of a Formula,*)
We define the height h of a prioritized formula as:

- $h(\top) = 0$ and $h(\bot) = 0$.
- If A is a variable $h(A) = 0$.
- $h(\neg A) = 1 + h(A)$.
- $h((A)) = h(A)$.
- $h(A \circ B) = max(h(A), h(B)) + 1$, if $\circ$ is one of the following operations: $\vee, \wedge, \Rightarrow, \Leftrightarrow$.

1. Show that this definition is ambiguous, that is, show that there exists at least one formula (by presenting one) that can have two different heights under this definition.
2. Give a new recuseive definition of height of a formula that does not have this problem.

Exercise 6 (Validity)
Give the truth table of the following formulas:

1. $p \Rightarrow (q \Rightarrow p)$.
2. $p \Rightarrow (q \Rightarrow r)$.
3. $(p \Rightarrow q) \Rightarrow (p \Rightarrow r)$.
4. $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$.

Indicate which formulas are valid, and which are equivalent.

Exercise 7 (Circular Reasoning)
Give the truth table of the following formulas: $a \Rightarrow b$, $b \Rightarrow c$, $c \Rightarrow a$, $a \Leftrightarrow b$, $b \Leftrightarrow c$, and $c \Leftrightarrow a$. Conclude that $(a \Rightarrow b) \wedge (b \Rightarrow c) \wedge (c \Rightarrow a) \equiv (a \Leftrightarrow b) \wedge (b \Leftrightarrow c) \wedge (c \Leftrightarrow a)$.

Exercise 8 (Equivalence)
Give the truth table of the following formulas:

1. $p \wedge (p \vee q)$.
2. $\neg p \wedge \neg q$.
3. $\neg(p \wedge q)$.
4. $\neg(p \vee q)$.
5. $p \vee (p \wedge q)$.
6. $\neg p \vee \neg q$.
7. p.

Indicate which formulas are equivalent.

Exercise 9 (Equivalence)
Give the truth table of the following formulas:

1. $(p \Rightarrow q) \wedge (q \Rightarrow p)$.
2. $p \Rightarrow q$.
3. $p \Leftrightarrow q$.
4. $\neg q \Rightarrow \neg p$.
5. $(p \wedge q) \vee (\neg p \wedge \neg q)$.

Indicate which formulas are equivalent.

Exercise 10 (Equivalence)
Which of the following formulas are equivalent to $p \Rightarrow q \vee r$?

1. $q \wedge \neg r \Rightarrow p$.
2. $p \wedge \neg r \Rightarrow q$.
3. $\neg q \wedge \neg r \Rightarrow \neg p$.
4. $q \vee \neg p \vee r$.

Exercise 11 (✎ Model for a set of formulas,*)
Show that an assignment is a model for a set of formulas if and only if it is a model for the conjunction of all the formulas in the set.

Exercise 12 (✎ Simplification Laws)
Prove that for any x, y

- $x \vee (x \wedge y) \equiv x$.
- $x \wedge (x \vee y) \equiv x$.
- $x \vee (\neg x \wedge y) \equiv x \vee y$.

Exercise 13 (✎ Simplification of Formula,*)
Show by simplification that the following formula is a tautology.

$$(a + b) \cdot (\bar{b} + c) \Rightarrow (a + c)$$

Exercise 14 (Models and Normal Forms)
Let A be the following formula:

$$(((a \Rightarrow \neg b) \Leftrightarrow \neg c) \wedge (c \vee d)) \wedge (a \Leftrightarrow d).$$

1. Is A a tautology? (justify)
2. Is A a contradiction ? (justify)
3. Give the conjunctive normal form for A.
4. Give the disjunctive normal form for A.

Exercise 15 (✎ Natural induction,*)
Show that if a formula contains only one variable and the operations $\vee$ and $\wedge$ (no negation), then it is equivalent to a formula of size 0.

Exercise 16 (Boolean Algebra)
Using truth tables, determine if the operations $\Rightarrow, \Leftrightarrow$ are commutative, associative, idempotent, transitive.

Exercise 17 (Boolean Algebra)
Show that a formula with only one variable (say, that variable is p) is equivalent to either 0, 1, p, or $\neg p$.

Exercise 18 (Boolean Algebra)
Write 16 formulas such that any formula with two variables, say p and q, is equivalent to one of those 16 formulas.

Exercise 19 (Boolean Function)
Determine the number of distinct (non-equivalent) boolean functions with k arguments (express your answer as a function of k).

Exercise 20 (Consequence)
During an inquiry, adjutant Tinet makes the following reasoning:

- If the murder occurred during the day, then the murderer is a friend of the victim
- However, the murder occurred at night

Therefore, the murderer is not a friend of the victim. Is adjutant Tinet correct? To determine this, proceed in three steps:

1. Formalize the facts
2. Formalize the reasoning that allows to arrive to the conclusion from the hypotheses
3. Determine if the reasoning is correct.

Exercise 21 (Consequence)
Pinnochio, Quasimodo and Romeo are singing in a choir. They decide that:

1. If Pinocchio does not sing, then Quasimodo will.
2. If Quasimodo sings, then so will Pinocchio and Romeo.
3. If Romeo sings, then at least one of Quasimodo or Pinocchio will not.

Determine whether Pinocchio will sing. Justify your answer by formalizing the reasoning.

Exercise 22 (Consequence)
Formalize the following statements using logic connectors.

(a) If Peter went home, then John went to the cinema.
(b) Mary went to the library or Peter went home.
(c) If John went to the cinema, then Mary went to the library or Peter went home.
(d) Mary is not at the library and John went to the cinema.
(e) Peter went home.

Is the last statement a consequence of the preceding ones?

Exercise 23 (Normal Forms)
For each of the following formulas, write the equivalent disjunctive normal form, and prove whether or not it is satisfiable (by giving a model for the formula if possible).

- $\neg(a \Leftrightarrow b) \vee (b \wedge c) \Rightarrow c$.
- $(a \Rightarrow b) \wedge (b \Rightarrow \neg a) \wedge (\neg a \Rightarrow b) \wedge (b \Rightarrow a)$.

Exercise 24 (Normal Forms)
Let A be the formula $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$. Find the formula in disjunctive normal form equivalent to $\neg A$. Is A valid?

Exercise 25 (☞ Normal Forms,)**
We come back to the proofs regarding the transformation of formulas into normal forms.

1. Show that the repeated application of the elimination of equivalence, elimination of implications and displacement of negations on a formula yields a formula in normal form.
2. Show also regardless of the order in which the elinimation of equivalence, elimination of implications and displacement of negations are used, the algorithm for transforming a formula in normal form will terminate.

Exercise 26 (Consequence and normal forms)
Adjutant Tinet is on a new inquiry. His hypotheses are as follows:

- If John did not meet Peter the other night, then Peter is the murderer or John is a liar.
- If Peter is not the murderer, then John did not meet Peter the other night and the crime occurred after midnight.
- If the crime occurred after midnight, then Peter is the murderer or John is a liar.

He concludes that Peter is the murderer. Is his reasoning correct? Give your answer by constructing the conjunction of the hypotheses and the *negation of the conclusion*, and by putting this disjunction in conjunctive normal form. Recall that a reasoning is incorrect if and only if one of the monomials does not have any complementary litterals: this monomial gives a model of the hypotheses that is a counter-model of the conclusion.

Exercise 27 (Formalization, sum of monomials*)
On an isolated island, the natives are split into two tribes: the Tame, and the Lame. The Tame always tell the truth, the Lame always lie. We meet two natives: Aha and Beeby[1]

(a) Aha says: " At least one of us is a Lame ". Can we deduce Aha and Beeby's tribe?
(b) Aha says: " At most one of us is a Lame ". Can we deduce Aha and Beeby's tribe?
(c) Aha says: " Both of us are in the same tribe ". Can we deduce Aha and Beeby's tribe?

Exercise 28 (Complete and incomplete set of connectors,)**
A set of constants and connectors is called *complete* if any boolean function can be expressed with these connectors. Theorem 6.4 page 25 shows thaat the set $\{0, 1, +, -, \cdot\}$ is complete.

1. Show that the set $\{+, -\}$ is complete.
   Hint: it suffices to show that $\{0, 1, \cdot\}$ can be defined only with $+$ and $-$.
2. Show that the set $\{0, \Rightarrow\}$ is complete.
3. Let $|$ be the following operation: $x \mid y$ is true if and only if neither x nor y is true, that is $x \mid y$ is true only if $x = 0$ and $y = 0$. This operation is also called *NOR* (for "not OR") because it is the negation of the "OR" ($\vee$) connector.
   Show that $\{|\}$ is complete.
4. (**) Show that the set $\{0, 1, +\cdot\}$ is not complete.
   Hint: boolean function that are defined using only these operations have a property that is not true of all boolean functions.
   We say that a boolean function is *monotone* (or *monotonic*) if whenever $a_1 \leq b_1, \ldots, a_n \leq b_n$, then $f(a_1, \ldots, a_n) \leq f(b_1, \leq, b_n)$.
   Show that every boolean function defined only with $\{0, 1, +, \cdot\}$ is monotone.
   Give a boolean function that is not monotone. Conclude that $\{0, 1, +, \cdot\}$ is not complete.

[1] This problem comes from a book by Raymond M. Smullyan : " What is the Name of this Book ", which describes many other amusing problems with these natives.

5. (*) Is the set $\{1, \Rightarrow\}$ complete?
6. (**) Is the set $\{0, \Leftrightarrow\}$ complete?

Exercise 29 (↬ Duality, recurrence,)**
Consider formulas that only contains the operations $0, 1, \neg, \wedge, \vee$. The dual of one of these formulas is obtained by turning conjunctions into disjunctions, 0 into 1 and vice versa. We write A^* to denote the dual of formula A.

1. Define A^* y recurrence on these formulas.
2. Show that if 2 formulas are equivalent, then so are their duals.
   Hint: let v be an assignment, and let $\overline{v}$ be the assignment complementary to v (0 becomes 1, 1 becomes 0).
   (a) Show that for every formula A above, $[A^*]_v = \overline{[A]_{\overline{v}}}$.
   (b) Deduce that if 2 such formulas are equivalent, then so are their duals.
3. Deduce from the preceding question that if a formula is valid, its dual is contradictory.

Exercise 30 (Canonical Form,*)**
Let $x_{i(0 \leq i \leq n)}$ be a list of n distinct variables. In this exercise, all the variables used in formulas are in this list. To any formula A, we associate a boolean function $f(x_1, \ldots, x_n) = A$. This means that for $a_1, \ldots a_n \in \mathbb{B}$, the value of $f(a_1, \ldots, a_n)$ is the value taken by A when $x_1 = a_1, \ldots, x_n = a_n$. We say that two formulas A and B are in the same class if A and B are equivalent.

1. How many classes of formulas (with our n variables) are there?
2. We denote by $\oplus$ le exclusive-or operation. We have $x \oplus y = 1$ if and only if $x = 1$ or $y = 1$, but not both (hence the name exclusive-or). We can also define this operation as $x \oplus y = \neg(x \Leftrightarrow y)$. It should be clear that this operation is commutative, associative, that $\wedge$ distributes on $\oplus$ and that 0 is the neutral element for $\oplus$. Express negation, disjunction, implication and equivalence as an exclusive-or sum of 1's and conjunctions or variables (i.e. like a formula in disjunctive normal form, except that the $\oplus$ is used instead of $\vee$).
3. Show that every boolean formula is equivalent to an exclusive-or sum of 1's and conjunctions or variables.
4. As a convention, the product of zero variables is considered to be 1 and the exclusive-or sum of zero variables is 0. Show that every boolean formula is equivalent to an exclusive-or sum of products such that not two monomials have the same set of variables.
5. Fix an arbitrary ordering of the n variables used in the formulas (when variables are identifiers, we often use the lexicographic order). An exclusive-or sum of products of variables is in Reed-Muller Canonical Form if and only if:
   - the products do not contain any repeated variables, and inside each products, the variables are ordered from left to right in increasing order,
   - the products are orderes from left to right in inverse lexicographic order deduced from the ordering of the variables.

   Show that every boolean formula is equivalent to a formula in Reed-Muller Cacnonical Form. Deduce that the expression of a formula in Reed-Muller Canonical form is unique.

Exercises for Chapter I.2

Exercise 31 (Resolvent)
Through resolution, we have:

$$\frac{a+b \quad \overline{a}+\overline{b}}{b+\overline{b}}$$

Show that: $b+\overline{b} \not\models (a+b).(\overline{a}+\overline{b})$.

Exercise 32 (Resolvent)
We recall that two clauses are equal if and only if they have the sets of literals.

- Are the clauses $p+q+\overline{r}+q+p+s+q+\overline{r}$ and $s+q+\overline{r}+p$ equal?
- Are the clauses $p+q+r+p$ and $q+r+q+r+q+r$ equal? Is one included in the other? Is one the consequence of the other? Are they equivalent?
- Give all the resolvents of the clauses $a+\overline{b}+c$ and $a+b+\overline{c}$. Are these resolvents valid?

Exercise 33 (Proof)
The following sets of formulas are unsatisfiable.

- $\{a, a \Rightarrow b, \overline{b}\}$.
- $\{a+b, \overline{a}+c, \overline{a}+\overline{d}, d+\overline{c}, \overline{b}+a\}$.
- $\{a+b+c, \overline{a}+b, \overline{b}+c, \overline{c}+a, \overline{a}+\overline{b}+\overline{c}\}$.

Prove that fact using resolution.

Exercise 34 (Formalization and Resolution,*)
We note that: " x unless y " can be formalized as $\overline{x \Leftrightarrow y}$. In a haunted house, ghosts show their presence in two ways: **obscene chants** and **sardonic laughs**. However, we can influence their behaviour by playing the organ or burning incense. Given the following:

(i) The ghosts don't do obscene chants unless we play the organ while the ghosts are not laughing.
(ii) If we burn incense, the ghosts laugh if and only if they are also chanting.
(iii) At the moment, the ghosts are chanting, but not laughing.

And our conclusion:

(iv) At the moment, we are playing the organ, but not burning incense.

We define the following:

- c: the ghosts are chanting.
- o: we are playing the organ.
- l: the ghosts are laughing.
- i: we are burning incense.

1. Transform the expression $\overline{x \Leftrightarrow y}$ into a product of clauses.
2. Formalize the hypotheses and the negation of the conclusion into a product of clauses.
3. Using resolution, prove that the reasoning is correct.

In other words, transform the conjunction of the hypotheses and the negation of the conclusion into a product of clauses and deduce the empty clause.

Exercise 35 (Proof,*)
Show, using a proof by resolution, the correctness of the following reasoning:

$$r + q \Rightarrow t, t.q \Rightarrow r, q \models t \Leftrightarrow r.$$

Exercise 36 (Formalization and Proof,*)
Show, using resolution, that the following reasoning is correct:

- The weather is nice unless it is snowing.
- It is raining unless it is snowing.
- The weather is nice or it is raining.
- Hence, it is not snowing.

Exercise 37 (↬ Defining a Clause)
A clause is either the empty clause, or a (non-empty) disjunction of literals. Give a inductive definition of a clause and define by recurrence the function s such that $s(C)$ is the set of literals in clause C.

Exercise 38 (↬ Proof)
Prove the property 2.1.14 page 44 on the monotony and composition.

Exercise 39 (↬ Property of Resolution)
Prove property 2.1.8 page 43.

Exercise 40 (↬ Resolution does not add literals,*)
Let Γ be a set of clauses. A literal in Γ is a literal that appears in one of the clauses of Γ. Show that any clause deduced from Γ only contains literals in Γ.

Exercise 41 (Formalization and resolution)
Consider the following hypotheses:

1. If Peter misses his tournament then Peter will be depressed.
2. If the weather is nice then Peter will go to the swimming pool.
3. If Peter does not go to the swimming pool he will be depressed.
4. When at the swimming pool, Peter does not train.
5. Peter will miss his tournament if he does not train.

We would like to demonstrate that under such hypotheses, the following conclusion can be derived:

6. Peter will be depressed.

You will proceed as follows:

- Formalize the hypotheses and the negation of the conclusion.
- Deduce a set of clauses equivalent to your formal statements.

- Show that the conclusion can be correctly derived from the hypotheses, by proving using the resolution method that the set of clauses is contradictory.

Exercise 42 (Formalisation and resolution)

The Beatles were a rock band formed in the 60's in Liverpool. It was composed of four boys: Ringo Starr, Paul MacCartney, John Lennon and Georges Harrison. It has been quite hard to decide who would play which instrument, as the following extract from their discussion will show:

Paul said: "If Ringo doesn't play the guitar, then I will play the bass guitar and John the guitar."

Georges said: "I will play the guitar if, and only if, John plays it too."

John said: "If Paul plays the bass guitar, then Georges will play the guitar."

Ringo said: "I will play the drums and therefore, not the guitar."

After their talk, they settled on the following power four:

- Ringo would play the drums,
- Paul would play the bass guitar
- John and Georges would both play the guitar.

We will now show that every member of the band is happy with this conclusion.

1. Formalise the 4 hypotheses and the conclusion using the following propositional variables:
   - RD: "Ringo plays the drums",
   - RG: "Ringo plays the guitar",
   - PB: "Paul plays the bass guitar",
   - JG: "John plays the guitar",
   - GG: "Georges plays the guitar".
2. Transform the hypotheses and the negation of the conclusion into clauses.
3. Prove, using resolution, that the conclusion is a consequence of the hypotheses.

Exercise 43 (Reduction, X1603 Andrews)
Soit l'ensemble de clauses :

$$\{p+q, \overline{p}+r+\overline{q}+p, p+\overline{r}, q+\overline{p}+\overline{q}, q+\overline{r}+p, r+q+\overline{p}+\overline{r}, \overline{r}+q\}.$$

1. Reduce this set (reduction is defined in 2.1.4 page 48).
2. Determine whether this set is satisfiable using resolution.

Exercise 44 (DPLL)
Consider the following set of clauses:

$$\{\overline{a}+\overline{b}+\overline{f}, a+b+f, e+\overline{a}, \overline{a}+\overline{b}, \overline{a}+c, d+a+\overline{d}, a+b, \overline{a}+\overline{c}+\overline{d}, d\}.$$

- Use algorithm Algo_DPLL on this set of clauses.
- Conclude whether or not it is satisfiable.
- If it is satisfiable, give a model obtained from the trace of the algorithm.

In the tree of calls, identify the steps as follows:

- Elimination of valid clauses (VAL).
- Reduction (RED).
- Pure literal elimination (PLE).
- Unit resolution (UR).

In addition, note the assignments made at each step of the algorithm so that the model can easily be recovered.

Exercise 45 (DPLL)
Use algorithm Algo_DPLL to determine whether or not the following sets of clauses are satisfiable:

- $\{a+b+c+d+e+f, \overline{a}+b, \overline{b}+a, \overline{c}+d, \overline{d}+c, \overline{b}+\overline{c}, \overline{b}+c, b+\overline{c}, \overline{e}, \overline{f}\}$.
- $\{a+b+c+d+f, \overline{a}+b, \overline{b}+a, \overline{c}+d, \overline{d}+c\}$.
- $\{b+j+\overline{a}, a+j+\overline{b}, b+a+j, a+j, j+b, \overline{b}+\overline{j}, \overline{j}+b, j+s, \overline{s}+\overline{b}\}$.
- $\{a+\overline{c}+d, \overline{b}+c+f, b+\overline{e}+f, \overline{c}+e+\overline{f}, e+f, c+d, \overline{a}, \overline{e}+\overline{f}\}$.

Give a trace of the algorithm.

Exercise 46 (Complete Strategy)
Let Γ be the following product of clauses:

$$(a+b+\overline{c}).(b+\overline{c}).(c+\overline{c}).(b+c).(\overline{a}+\overline{b}+c).(a+\overline{b}+c).$$

Determine using the complete resolution strategy whether Γ is unsatisfiable, or if it has a model. Give a trace of the algorithm. Give the proof(s) you obtained. If Γ has a model, show it.

Exercise 47 (Complete Strategy)
Consider the following set of clauses:

$$\{p+q, \overline{p}+s, \overline{s}+t, \overline{t}, \overline{q}+r, \overline{r}, \overline{r}+p+t, q+z+\overline{z}, \overline{q}+r+s\}.$$

Use the complete strategy on this set of clauses and determine whether it is satisfiable or not.

Exercise 48 (Complete Strategy)
Consider the function f such that $f(x,y,z)=0$ if and only if there is an even number of arguments with value 1. Express f as a product of clauses using the method described in subsection 1.6 page 32, then simplify f using the complete strategy.

Exercise 49 (✍ Proof)
Prove Lemma 2.3.2 page 55.

Exercise 50 (✍ Proof)
Prove Lemma 2.3.5 page 55.

Exercise 51 (From SAT to 3-SAT,*)**
SAT (short for Satisfiability) is a decision problem stated as follows: "given a set of clauses Γ, determine if Γ has a model". 3-SAT is a restriction of this problem in which each clause in Γ contains exactly three literals.

SAT is an NP-complete problem [5]. In this exercise, we study a polynomial-time reduction from SAT to 3-SAT, thereby showing that 3-SAT is also NP-complete[2].

A polynomial-time reduction from SAT to 3-SAT is a polynomial-time transformation of the set of clauses Γ into a set of clauses Γ' verifying the following two properties:

(a) all the clauses in Γ' contain three distinct literals.
(b) Γ has a model if and only if Γ' has a model.

Usually, such reductions also satisfy the following additional property:

(c) Any model for Γ' is also a model for Γ.

The goal of this exercise is to show that the polynomial-time transformation below verify these three properties. Note that the transformation potentially introduces new variables.

Let $\Gamma = \{c_1, \ldots, c_m\}$ be a set of clauses. The reduction consists in replacing every clause $c_i = z_1 + \ldots + z_k$ of Γ (where each z_i is a literal) by a set of clauses C'_i constructed as follows, depending on the value of k:

$k = 1$ **:** $C'_i = \{z_1 + y_1 + y_2,\ z_1 + y_1 + \overline{y_2},\ z_1 + \overline{y_1} + y_2,\ z_1 + \overline{y_1} + \overline{y_2}\}$ where y_1 and y_2 are new variables (not present anywhere else).

$k = 2$ **:** $C'_i = \{z_1 + z_2 + y_1,\ z_1 + z_2 + \overline{y_1}\}$ where y_1 is a new variable.

$k = 3$ **:** $C'_i = \{c_i\}$.

$k > 3$ **:** $C'_i = \{z_1 + z_2 + y_1,\ \overline{y_1} + z_3 + y_2,\ \overline{y_2} + z_4 + y_3,\ \overline{y_3} + z_5 + y_4,\ \ldots,\ \overline{y_{k-3}} + z_{k-1} + z_k\}$ where $y_1, \ldots, y_{k-3}$ are new variables.

Finally, $\Gamma' = \bigcup_{i=1}^{m} C'_i$.

By construction, the reduction certainly verify property (a). The answers to the following questions allow us to prove properties (b) and (c):

1. Show (without a truth table) that any assignment will give the same value to c_i as to the conjunction of all the clauses in C'_i when c_i consists of only one literal.
2. Show (without a truth table) that any assignment will give the same value to c_i as to the conjunction of all the clauses in C'_i when c_i consists of two literal.
3. Let c_i be a clause with more than 3 literals. Show that if c_i has a model, then C'_i also has a model.
4. Let c_i be a clause with more than 3 literals. Show that any model for C'_i is also a model for c_i.
5. The reduction above maintains satisfiability. Show that the reduction does not maintain the meaning of the set of formulas. That is, the conjunction of all the clauses in Γ is not equivalent to the conjunction of all the formulas in Γ'.

[2] We note that the k-SAT is not NP-complete for any value of k: 2-SAT can be solved in polynomial time.

Exercises for Chapter I.3

Exercise 52 (Proof Sketch)
The following sequence of lines is not a proof sketch. Find the smallest number i such that lines 1 to $i-1$ is a proof draft, but lines 1 to i is not. Give the context of lines 1 to $i-1$.

context	number	proof
	1	`Suppose` a
	2	`Suppose` b
	3	c
	4	`Thus` d
	5	`Suppose` e
	6	f
	7	`Thus` g
	8	h
	9	i
	10	`Thus` j
	11	`Thus` k
	12	l

Exercise 53 (Proof of Formulas Requiring Special Rules)
Give a proof of the following formulas:

- $a \Rightarrow \neg\neg a$,
- $\neg\neg a \Rightarrow a$,
- $a \Leftrightarrow \neg\neg a$,
- $a \vee \neg a$.

Exercise 54 (Simple Proofs of Formulas)
Give a proof of the following formulas:

- $a \Rightarrow c$ in environment $a \Rightarrow b,\ b \Rightarrow c$.
- $(a \Rightarrow b) \wedge (b \Rightarrow c) \Rightarrow (a \Rightarrow c)$.
- $(a \Rightarrow b) \Rightarrow ((b \Rightarrow c) \Rightarrow (a \Rightarrow c))$.

Exercise 55 (☞ Abbreviations)
Let A be a formula and $ufld(A)$ be the formula obtained by replacing in A the negations and equivalences (in abbreviated form) by their definition. $ufld(A)$ is the formula obtained by "unfolding", hence the name $ufld$ chosen for the unfolding function.

- Define the function $ufld$ by recurrence.
- Show that the formulas A and $ufld(A)$ are equivalent.
- Deduce that two formulas, equal up to abbreviations, are equivalent.

Exercise 56 (Proof of Formulas)
Give a proof of the following formulas:

1. $a \Rightarrow (b \Rightarrow a)$.
2. $a \wedge b \Rightarrow a$.

3. $\neg a \Rightarrow (a \Rightarrow b)$.
4. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$.
5. $a \wedge b \Rightarrow b \wedge a$.
6. $a \vee b \Rightarrow b \vee a$.
7. $(a \Rightarrow (b \Rightarrow c)) \Rightarrow (a \wedge b \Rightarrow c)$.
8. $(a \wedge b \Rightarrow c) \Rightarrow (a \Rightarrow (b \Rightarrow c))$.
9. $(a \Rightarrow b) \wedge (c \Rightarrow d) \Rightarrow (a \wedge c \Rightarrow b \wedge d)$.

Exercise 57 (About implication)
Give a proof of the following formulas:

1. $a \Rightarrow b$ in the environment $\neg a \vee b$.

2. $(\neg b \Rightarrow \neg a) \Rightarrow (a \Rightarrow b)$.

Exercise 58 (Boolean algebra)
Give a proof of the following formulas:

1. $\neg(a \wedge \neg a)$.

2. $a \vee a \Rightarrow a$.

3. $a \wedge a \Rightarrow a$.

4. $a \wedge (b \vee c) \Rightarrow a \wedge b \vee a \wedge c$.

5. $a \wedge b \vee a \wedge c \Rightarrow a \wedge (b \vee c)$.

6. $a \vee b \wedge c \Rightarrow (a \vee b) \wedge (a \vee c)$.

7. $(a \vee b) \wedge (a \vee c) \Rightarrow a \vee b \wedge c$.

Exercise 59 (Proof of Formulas with Environment)
Give a proof of the following formulas:

- P1 : $B \vee C$ in environment $\neg B \Rightarrow C$.
- P2 : $\neg B \vee C$ in environment $B \Rightarrow C$.
- P3 : $\neg B \vee \neg C$ in environment $\neg(B \wedge C)$.
- P4 : $\neg B$ in environment $\neg(B \vee C)$.
- P5 : $\neg C$ in the same environment $\neg(B \vee C)$.
- P6 : B in the same environment $\neg(B \Rightarrow C)$.
- P7 : $\neg C$ in the same environment $\neg(B \Rightarrow C)$.

Exercise 60 (Proof of Formulas)
Give a natural deduction proof of the following formulas:

1. $\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b)$.
2. $(\neg a \wedge \neg b) \Rightarrow \neg(a \vee b)$.
3. (*) $(\neg a \vee \neg b) \Rightarrow \neg(a \wedge b)$.
4. (**) $(a \vee b) \wedge (\neg a \vee b) \Rightarrow b$.
5. (***) $\neg(a \wedge b) \Rightarrow (\neg a \vee \neg b)$.

6. (***) $(a \vee b) \wedge (\neg b \vee c) \Rightarrow a \vee c$.

Exercise 61 (Midterm 2011)
Prove the following formulae using natural deduction :

- $(p \Rightarrow \bot) \wedge (\neg p \wedge q \Rightarrow \bot) \Rightarrow \neg q$
- $((p \vee q) \vee (p \vee r)) \wedge \neg p \Rightarrow q \vee r$
- (*) $(p \Rightarrow \bot) \vee (p \wedge q \Rightarrow \bot) \Rightarrow \neg q \vee \neg p$

Exercise 62 (Midterm 2013)
Give a proof of the following formulae using the natural deduction in the table format :

- $(p \vee q) \Rightarrow (\neg p \wedge \neg q) \Rightarrow r$.
- $((p \Rightarrow q) \wedge (q \Rightarrow r)) \wedge \neg r \Rightarrow \neg p$.
- $(p \Rightarrow q) \Rightarrow ((p \wedge q) \vee \neg p)$.

Exercise 63 (Questions from various midterm exams)

1. Prove the following formula using natural deduction: $(p \vee q) \wedge (p \Rightarrow r) \Rightarrow q \vee r$
2. We consider the hypotheses:
   - (H1) : If Peter is tall, then John isn't Peter's son.
   - (H2) : If Peter isn't tall, then John is Peter's son.
   - (H3) : If John is Peter's son, then Mary is John's sister.

   Formalize and prove that from these we can deduce conclusion (C) : "Mary is John's sister or Peter is tall or both" using natural deduction.

Exercises for Chapter II.4

Exercise 64 (Structure and free variables)

For each formula hereafter, indicate its structure and its free variables.

1. $\forall x(P(x) \Rightarrow \exists y Q(x,y))$.
2. $\forall a \forall b(b \neq 0 \Rightarrow \exists q \exists r(a = b * q + r \wedge r < b))$[3].
3. $Even(x) \Leftrightarrow \exists y(x = 2 * y)$.
4. $Divides(x,y) \Leftrightarrow \exists z(y = z * x)$.
5. $Prime(x) \Leftrightarrow \forall y(Divides(y,x) \Rightarrow y = 1 \vee y = x)$.

Exercise 65 (Formalization, function and relation symbols)
Consider $\Sigma = \{b^{r2}, u^{r2}, c^{r2}, o^{r2}, r^{f0}, f^{f1}\}$, a signature with the following meaning:

- $b(x,y) :=$ x is one of y's brothers.
- $u(x,y) :=$ x is one of y's uncles.
- $c(x,y) :=$ x is one of y's cousins.
- $o(x,y) :=$ x is older than y.
- r is Robert's nickname.
- $f(x)$ returns x's father.

Formalize the following sentences in first order logic using signature Σ.

1. Every brother of Robert's father is Robert's uncle.
2. If the fathers of two children are brothers, then these children are cousins.
3. One of Robert's cousins is younger than one of Robert's brothers.

Translate in English the following logical propositions:

1. $\neg \exists x\ o(x, p(x))$
2. $\forall x \forall y(f(f(x)) = f(f(y)) \Rightarrow c(x,y))$
3. $\forall x \exists y(b(x,y) \wedge o(y,x))$
4. $\exists x \exists y(b(x,y) \wedge \neg(f(x) = f(y)))$

Exercise 66 (Formalization)
Consider signature $\Sigma = \{g^{f0}, f^{f0}, P^{r2}, W^{r2}\}$, whose symbols have the following meaning:

- $g :=$ Germany's team.
- $f :=$ France's team.
- $P(x,y) :=$ x played against y.
- $W(x,y) :=$ x won against y.

Formalize the following sentences in first order logic using siganture Σ.

1. France's team won a match and lost a match.
2. France's and Germany's team tied.
3. A team won all its matches.

[3] In order to respect the usual notation for the Euclidean Algorithm, we exceptionally use a, b, q and r as variable names

4. No team lost all its matches.
5. Consider the following assertion: "All the teams who played against a team that won all its matches won at least one match". Which of the following formulas formalize this assertion, and which are equivalent?
   (a) $\forall x \exists y\, (P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z)) \Rightarrow \exists v W(x,v))$.
   (b) $\forall x(\exists y\, (P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z))) \Rightarrow \exists v W(x,v))$.
   (c) $\exists x(\forall y\, (P(x,y) \Rightarrow W(x,y)) \Rightarrow \forall z(P(x,z) \Rightarrow \exists v W(x,v)))$.
   (d) $\forall x \forall y\, (P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z)) \Rightarrow \exists v W(x,v))$.
   (e) $\forall x(\forall y\, (P(x,y) \wedge \forall z(P(y,z) \Rightarrow W(y,z))) \Rightarrow \exists v W(x,v))$.

Exercise 67 (Formalization,)**
We define constants s for Serge, t for Tobby, $L(x,y)$ for x loves y, D(x) for x is a dog, $P(x)$ for x is a pet, $K(x)$ for x is a kid, $B(x)$ for x is a bird and $S(x,y)$ for x is scared of y. Describe the signature Σ associated for these symbols, and formalize the following statements in first order logic using that signature.

1. Dogs and birds are pets.
2. Tobby is a dog that loves kids.
3. Birds do not like dogs.
4. Serge loves all pets except dogs.
5. All kids are not scared of dogs.
6. Some dogs love kids.
7. Some dogs love kids, and vice versa.
8. Kids love some dogs.

Exercise 68 (Evaluation of unary predicates)
Let I be the interpretation on domain $D = \{0,1\}$ with $P_I = \{0\}$ and $Q_I = \{1\}$.

1. Evaluate the following formulas with I: $\forall x P(x)$ and $\forall x(P(x) \vee Q(x))$.
2. Are the formulas $\forall x P(x) \vee \forall x Q(x)$ and $\forall x(P(x) \vee Q(x))$ equivalent?
3. Evaluate the following formulas with I: $\exists x P(x)$ and $\exists x(P(x) \wedge Q(x))$.
4. Are the formulas $\exists x P(x) \wedge \exists x Q(x)$ and $\exists x(P(x) \wedge Q(x))$ equivalent?
5. Evaluate the following formulas with I: $\forall x(P(x) \Rightarrow Q(x))$ and $\forall x P(x) \Rightarrow \forall x Q(x)$.
6. Are the formulas $\forall x(P(x) \Rightarrow Q(x))$ and $\forall x P(x) \Rightarrow \forall x Q(x)$ equivalent?

Exercise 69 (Interpretation,*)
Consider the following formulas:

1. $\forall x \exists y(y = x + 1)$.
2. $\exists y \forall x(y = x + 1)$.
3. $\forall x \exists y(y = x + 1) \Rightarrow \exists y \forall x(y = x + 1)$.
4. $\forall x \exists y(x = y + 1)$.
5. $\exists x \forall y(y = x + y)$.
6. $\exists x(x \neq 0 \wedge x + x = x)$.

and the following interpretations:

1. I1 is Boolean Algebra on $\{0,1\}$.
2. I2 is the usual arithmetic on natural numbers.
3. I3 is the usual arithmetic on rational numbers.

4. I4 is the Boolean Algebra on $\mathcal{P}(X)$ where constants 0 and 1 denote the sets $\emptyset$ and X, and addition represents set union.

Indicate if these interpretations are models or counter-models of the formulas above.

Exercise 70 (Unary predicats and equality)
Let I be the interpretation on the domain $D = \{0, 1, 2\}$ with $P_I = \{0, 1\}, Q_I = \{1, 2\}, R_I = \{\}$. Evaluate the following formulas with this interpretation:

1. $\exists x R(x)$.
2. $\forall x(P(x) \vee Q(x))$.
3. $\forall x(P(x) \Rightarrow Q(x))$.
4. $\forall x(R(x) \Rightarrow Q(x))$.
5. $\exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$[4].
6. $\exists x(P(x) \wedge Q(x) \wedge \forall y(P(y) \wedge Q(y) \Rightarrow x = y))$.

Exercise 71 (Evaluation, equality)
We use the unary function symbol f and the constant a. Let $x \neq y$ be a shorthand for $\neg(x = y)$. We define interpretations I_1, I_2 on domain $\{0, 1, 2\}$ as follows:
$a_{I_1} = a_{I_2} = 0$.

x	$f_{I_1}(x)$	$f_{I_2}(x)$
0	0	1
1	0	2
2	2	0

In interpretation I_1, then in I_2, evaluate the following formula:

1. $f(a) = a$.
2. $f(f(a)) = a$.
3. $f(f(f(a)) = a$.
4. $\exists x(f(x) = x)$, (f has a fixed point).
5. $\forall x(f(f(f(x))) = x)$.
6. $\forall y \exists x(f(x) = y)$, ($f$ is surjective, or onto).
7. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$, ($f$ is injective, or one-to-one.
8. $\neg \exists x \exists y(f(x) = f(y) \wedge x \neq y)$.

Exercise 72 (Formalization and evaluation)
We use the following notation:

- $P(x)$ means x passed his exam
- $Q(x, y)$ means x called y

Give the signature associated with these symbols, and formalize the following statements:

1. Someone failed the exam and was not called by anyone.
2. All those who passed the exam received a call.
3. Nobody called those who passed the exam.
4. All those cho called someone else called someone who passed the exam.

[4] This formula means that there is only one element satisfying P.

We define an interpretation with domain $D = \{0, 1, 2, 3\}$. Let Anatoli, Boris, Catarina and Denka be four constants with value 0, 1, 2, 3 respectively. Anatoli and Boris are boys, Catarina and Denka are girls. In our interpretation, only Boris and Catarina passed the exam, every boy called every girl, Denka called Boris, Catarina called Denka and no other calls were made. Evaluate the statements above in this interpretation.
Indication : to facilitate the evaluation, it might help to represent the interpretation pictorially, by circling people who passed the exam and drawing an arrow from x to y if x called y.

Exercise 73 (Expansion and counter-models)
Using the method of expansion, find counter-models to the following formulas:

1. $\exists x P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$.
2. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow \exists x Q(x)$.
3. $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\exists x P(x) \Rightarrow \forall x Q(x))$.
4. $(\exists x F(x) \Rightarrow \exists x G(x)) \Rightarrow \forall x(F(x) \Rightarrow G(x))$.
5. $\forall x \exists y R(x, y) \Rightarrow \exists x R(x, x)$.
6. $\forall x \forall y(R(x, y) \Rightarrow R(y, x)) \Rightarrow \forall x R(x, x)$.

Hint : it should be enough to build 1 or 2 expansions.

Exercise 74 (Incorrect reasoning)
Consider the following hypotheses:

1. $\exists x P(x)$.
2. $\exists x Q(x)$.
3. $\forall x(P(x) \wedge Q(x) \Rightarrow R(x))$.

Show that it is incorrect to deduce from these that $\exists x R(x)$.

Exercise 75 (Counter-model with a relation)
Construct counter-models for the following formulas, where F is a relation:

1. $\forall x \exists y(x = y) \Rightarrow \exists y \forall x(y = x)$.
2. $F(a) \wedge (a \neq b) \Rightarrow \neg F(b)$.
3. $\exists x \exists y(F(x) \wedge F(y) \wedge x \neq y) \Rightarrow \forall x F(x)$.
4. $\forall x \forall y(F(x, y) \Rightarrow x = y) \Rightarrow \exists x F(x, x)$.

Exercise 76 (Counter-model with a function)
Construct counter-models for the following formulas, where f is a function and P is a relation :

1. $\forall y \exists x(f(x) = y)$.
2. $\forall x \forall y(f(x) = f(y) \Rightarrow x = y)$.
3. $\exists x \forall y(f(y) = x)$.
4. $\forall x(P(x) \Rightarrow P(f(x))$.

Exercise 77 (Equivalences)
Prove that

1. $\neg \forall x \exists y P(x, y) \equiv \exists y \forall x \neg P(y, x)$.
2. $\exists x(P(x) \Rightarrow Q(x)) \equiv \forall x P(x) \Rightarrow \exists x Q(x)$.

3. The sentence "no sick person likes charlatans" was formalized in first order logic by two students as follows:
   - $\forall x \forall y((S(x) \wedge L(x,y)) \Rightarrow \neg C(y))$.
   - $\neg(\exists x(S(x) \wedge (\exists y(L(x,y) \wedge C(y)))))$.

   Show that both students are saying the same thing, that is, these two formulas are equivalent.

Exercise 78 (✒ , Proofs,*)
Prove the following two equivalences from Lemma 4.4.1:

- $\neg \exists x A \equiv \forall x \neg A$.
- $\exists x A \equiv \neg \forall x \neg A$.

Hint : use properties 1 and 2 from Lemma 4.4.1.

Exercise 79 (Proof)
We know that $(\forall x(A \wedge B)) \equiv (A \wedge (\forall x B))$ under the condition that x is not a free variable in A, as mentioned in paragraph 4.4.2. Show that this condition is necessary by giving an assignment that give different values to the formulas $\forall x(P(x) \wedge Q(x))$ and $P(x) \wedge (\forall x Q(x))$ when this condition is not met.

Exercises for Chapter II.5

Exercise 80 (Herbrand Universe)
Let Σ be the signature with constant a and function symbols f and g of arity one and two respectively.

- Give 5 distinct elements of the Herbrand universe of this signature.
- Give an inductive definition of this Herbrand universe.

Exercise 81 (Signature, Herbrand Universe and Base)
For each of the following sets of formulas,

- $\Gamma_1 = \{P(x) \vee Q(x) \vee R(x), \neg P(a), \neg Q(b), \neg R(c)\}$.
- $\Gamma_2 = \{P(x), \neg Q(x), \neg P(f(x)) \vee Q(f(x))\}$.
- $\Gamma_3 = \{P(x), Q(f(x)), \neg R(f(f(x))), \neg P(f(f(x))) \vee \neg Q(x) \vee R(f(x))\}$.

1. Give the signature, and the corresponding Herbrand universe and base.
2. Prove whether or not their universal closure has a model.

Exercise 82 (Herbrand's method)
Use Herbrand's method in order to prove that the following set of formulae is unsatisfiable:

1. $\forall x R(x, f(x))$
2. $\forall x \forall y (\overline{S(x,y)} \vee R(x,y) \vee R(y,x))$
3. $\forall x \forall y (S(x,y) \vee \overline{R(x,y)})$
4. $\forall x \forall y (S(x,y) \vee \overline{R(y,x)})$
5. $\forall y \overline{S(y,a)}$

Exercise 83 (Herbrand's method,*)
Use Herbrand's method in order to prove that the following set of formulae is unsatisfiable:

1. $\forall x (Q(x) \vee \neg P(f(x)))$
2. $\forall y (Q(y) \Rightarrow R(y))$
3. $\forall z (\neg P(z) \Rightarrow Q(z) \vee R(z))$
4. $\forall u \neg R(u)$

In particular, you will have to transform your set of unsatisfiable instances into an equivalent set of clauses. Then, you will prove that it is contradictory with a proof in propositional resolution.

Exercise 84 (Herbrand's method,*)
Let Γ be the following set of formulae:

1. $\overline{S(x,y)} + \overline{M(z,x)} + M(z,y)$
2. $S(x,y) + M(f(x,y),x)$
3. $S(x,y) + \overline{M(f(x,y),y)}$
4. $S(c,a)$
5. $S(a,b)$
6. $\overline{S(c,b)}$

Find a finite unsatisfiable set of closed instances of these formulae.

This enables us to deduce something about a set of first-order formulae : which set and what ?

Exercise 85 (Herbrand Model,*)**
Let Δ be the following set of formulas:

1. $x < y \wedge y < z \Rightarrow x < z$.
2. $\neg(x < x)$.
3. $x < y \Rightarrow x < f(x,y) \wedge f(x,y) < y$.
4. $a < b$.

- Give a model of $\forall(\Delta)$.
- Does $\forall(\Delta)$ have a model on the Herbrand universe created from a, b, f ?

Exercise 86 (Skolemization)
Skolemize the following formulas (pay attention to the negations !) then transform them in clausal form.

1. $\neg(\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x)))$.
2. $\neg(\forall x \forall y \forall z(e(x,y) \wedge e(y,z) \Rightarrow \neg e(x,z)) \Rightarrow \neg \exists x \forall y\ e(x,y))$.
3. $\neg(\neg \forall x P(x) \vee \neg \forall x Q(x) \Rightarrow \neg(\forall x P(x) \wedge \forall x Q(x)))$.
4. $\forall x((\exists y P(x,y) \Rightarrow \exists x Q(x)) \wedge \exists y P(x,y) \wedge \neg \exists x Q(x))$.
5. $\neg(\exists x \forall y \forall z((P(y) \Rightarrow Q(z)) \Rightarrow (P(x) \Rightarrow Q(x))))$

Exercise 87 (Unification)
Are the following terms unifiable? If so, give their most general unifier, if not, justify your answer.

- $h(g(x), f(a,y), z)$ et $h(y, z, f(u,x))$.
- $h(g(x), f(a,y), z)$ et $h(y, z, f(u, g(x)))$.

Exercise 88 (Unification)
Give the most general unifier for each of the following terms if it exists.

1. $R(a, f(x)) = R(y, f(g(y, b)))$,
2. $R(y, f(x)) = R(x, f(g(y, b)))$,
3. $Q(y, f(a), f(x)) = Q(g(a, b), x, f(y))$, et
4. $Q(y, f(x), g(y, x)) = Q(x, f(y), g(f(a), y))$.

Exercise 89 (Unification)
Give the most general unifier for each of the following terms if it exists.

1. $pair(a, crypt(z, b))$ et $pair(x, y)$.
2. $pair(crypt(x, b), crypt(y, b))$ et $pair(crypt(a, b), z)$.
3. $crypt(pair(z, a), x)$ et $crypt(pair(y, crypt(x, b)), b)$.
4. $crypt(pair(a, z), x)$ et $crypt(pair(y, crypt(x, b)), b)$.
5. $f(x, y, g(a, a))$ et $f(g(y, y), z, z)$
6. $f(x, y, a)$ et $f(y, g(z, z), x)$

Exercise 90 (Unification with multiple solutions)
The equation $f(g(y), y) = f(u, z)$ has two most general unifiers (Recall: they are therefore equivalent). Give these two solutions.

Exercise 91 (Proof by resolution,*)
Show that the following formula is valid by transforming its *negation* in clausal form and by finding a contradictory set of instances of the clauses.

$$\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x)).$$

Exercise 92 (Proof by resolution,)**
Consider the following formulas:

1. $H_1 = \exists x P(x) \Rightarrow \forall x P(x)$.
2. $H_2 = \forall x(P(x) \vee Q(x))$.
3. $C = \exists x \neg Q(x) \Rightarrow \forall x P(x)$.

We want to show that C is the consequence of H_1 and H_2 using instantiation and resolution.

1. Transform the three formulas H_1, H_2, $\neg C$ in clausal form.
2. Find contradictory instances of the clauses obtained and show by propositional resolution that these instances are contradictory.
3. Give a direct proof of this contradiction using factorization, copy and binary resolution. It is possible that you only need the last rule.

Exercise 93 (Proof by resolution,)**

- Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:

  1. $P(f(x)) \vee \neg Q(y, a)$.
  2. $Q(a, a) \vee R(x, x, b) \vee S(a, b)$.
  3. $S(a, z) \vee \neg R(x, x, b)$.
  4. $\neg P(f(c)) \vee R(x, a, b)$.
  5. $\neg S(y, z) \vee \neg S(a, b)$.

- Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:

  1. $P(x)$.
  2. $\neg P(y) \vee Q(y, x)$.
  3. $\neg Q(x, a) \vee \neg Q(b, y) \vee \neg Q(b, a) \vee \neg P(f(y))$.

- Using a proof by factorization, copy and binary resolution, prove that the universal closure of the following set of clauses is unsatisfiable:

  1. $R(x, a)$.
  2. $R(y, b)$.
  3. $R(z, a + b)$.
  4. $\neg(b < x + b) \vee (a + b < a)$.
  5. $b < z$.
  6. $\neg(a + b < x) \vee \neg R(y, b)$.

  Recall : The symbol $+$ has a higher priority than the symbol $<$, which itself has a higher priority than binary connectors.

Exercise 94 (Unification, resolution)
Let $\Gamma_1 = \{P(x, f(x, b), u), \neg P(g(a), z, h(z))\}$ and $\Gamma_2 = \{P(x, f(x, b), u), \neg P(g(z), z, h(z))\}$ Are the sets $\forall(\Gamma_1)$ and $\forall(\Gamma_2)$ satisfiable or unsatisfiable?

Exercise 95 (Skolemization and First Order Resolution)
The goal of this exercise is to prove the following syllogism:

$$\forall x(man(x) \Rightarrow mortal(x)) \wedge man(Socrates) \Rightarrow mortal(Socrates)$$

- Skolemize the **negation** of the syllogism.
- Transform the obtained Skolem form in clausal form.
- Demonstrate by instantiation that the **negation** of the syllogism is a contradiction.
- Demonstrate using factorization, copy and binary resolution that the negation of the syllogism is a contradiction.

Exercise 96 (Clausal Form and Resolution,)**
Consider the following formulas:

1. $A_1 = \exists u \forall v (P(u) \wedge (R(v) \Rightarrow Q(u,v)))$.
2. $A_2 = \forall u \forall v (\neg P(u) \vee \neg S(v) \vee \neg Q(u,v))$.
3. $A_3 = \exists v (R(v) \wedge S(v))$.

Show that the set containing these three formulas is contradictory using resolution:

1. Transorm A_1, A_2, A_3 in clausal form.
2. Find contradictory instances of the clauses you obtained and show this contradiction using propositional resolution.
3. Make a direct proof of $\perp$ using factorization, copy and binary resolution.

Exercise 97 (↬ Coherence of First Order Resolution)
Prove the coherence of first order resolution, that is:

Let Γ be a set of clauses and C a clause. If $\Gamma \vdash_{1fcb} C$ then $\forall(\Gamma) \models \forall(C)$.

Exercise 98 (Clausal Form and Resolution,)**
Consider the following formulas:

1. $E_1 = \forall x (x = x)$.
2. $E_2 = \forall x \forall y (x = y \Rightarrow y = x)$.
3. $E_3 = \forall x \forall y \forall z \forall t (x = y \wedge z = t \Rightarrow (x \in z \Rightarrow y \in t))$.
4. $C = \forall y \forall z (y = z \Rightarrow \forall x (x \in y \Leftrightarrow x \in z))$.

E_1, E_2, E_3 are axioms of equality: E_1 says equality is reflexive, E_2 says it is symmetric, E_3 says it is a congruence relative to $\in$. We do not use transitivity in this exercise. C says that two sets are equal if they have the same elements. We note that in this exercise, the symbol '=' is not treated as usual, as the symbol with fixed meaning describing the identity relation. The restriction on the appearance of the equality symbol in formulas applies only when it is considered as a symbol with fixed meaning. Show that C is a consequence of the other axioms on equality:

1. Put the formulas E_1, E_2, E_3, $\neg C$ in clausal form.
2. Find contradictory instances of the clauses you obtained.
3. Show a direct proof of $\perp$ using factorization, copy and binary resolution.

Exercise 99 (Clausal Form and Resolution,)**
Consider the following formulas:

1. $H_1 = \forall u (\exists v R(u,v) \Rightarrow R(u, f(u)))$.
2. $H_2 = \forall u \exists v R(u,v)$.
3. $H_3 = \exists u R(f(f(u)), u)$.
4. $C = \exists u \exists v \exists w (R(u,v) \wedge R(v,w) \wedge R(w,u))$.

Show that C is the consequence of H_1, H_2, H_3.

1. Transform the formulas H_1, H_2, H_3, $\neg C$ in clausal form.
2. Find contradictory instances of the clauses you obtained, and show the contradiction using propositional resolution.
3. Prove this contradiction directly using factorization, copy and binary resolution.

Exercise 100 (Clausal Form and Resolution,)**
Let $F(x, y)$ be a formalization of " x is y's father ". Let $A(x, y)$ be a formalization of " x is y's ancestor ". Consider the following formulas:

1. $H_1 = \forall x \exists y F(x, y)$.
2. $H_2 = \forall x \forall y \forall z (F(x, y) \wedge F(y, z) \Rightarrow A(x, z))$.
3. $C = \forall x \exists y A(x, y)$.

Show that C is the consequence of H_1, H_2 by transforming $H_1, H_2, \neg C$ into clausal form, and deriving the empty clause using factorization, copy and binary resolution.

Exercise 101 (Bernays-Schonfinkel Formula,*)**
A BS (Bernays-Schonfinkel) formulais a closed formula *without* any function symbols, of the form $\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_p\ B$, where B does not have any quantifiers. Show how to decide the satisfiability of such a formula.

Exercises for Chapter II.6

Exercise 102 (Natural Deduction)
Prove the following formulas using natural deduction:

1. The famous syllogism, "All men are mortal, Socrates is a man, therefore Socrates is mortal", which we formalize as $\forall x(H(x) \Rightarrow M(x)) \wedge H(socrates) \Rightarrow M(socrates)$.
2. $\forall x P(x) \Rightarrow \exists y P(y)$.
3. $\forall x P(x) \Rightarrow \exists x P(x)$.
4. $\forall x(P(x) \wedge Q(x)) \Rightarrow \forall x P(x) \wedge \forall x Q(x)$. Note that example 6.1.2 gives the proof of the reciprocal.
5. $\exists x(P(x) \vee Q(x)) \Rightarrow \exists x P(x) \vee \exists x Q(x)$.
6. $\forall x(P(x) \Rightarrow Q(x)) \wedge \exists x P(x) \Rightarrow \exists x Q(x)$.
7. $\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists x(P(x) \vee Q(x))$. (**)
8. $\forall x(P(x) \Rightarrow Q(x)) \wedge \forall x(Q(x) \Rightarrow R(x)) \Rightarrow \forall x(P(x) \Rightarrow R(x))$.
9. $\forall x\ (P(x) \Rightarrow Q(x)) \wedge \exists x\ \neg Q(x) \Rightarrow \exists x\ \neg P(x)$.

In this exercise, the formulas $P(x)$ and $Q(x)$ can be replaced by *any* other formulas.

Exercise 103 (Natural Deduction)
Prove the following formulas:

1. $\forall x \forall y P(x, y) \Rightarrow \forall y \forall x P(x, y)$.
2. $\exists x \exists y P(x, y) \Rightarrow \exists y \exists x P(x, y)$.
3. $\exists x \forall y P(x, y) \Rightarrow \forall y \exists x P(x, y)$.
4. $\forall x(Q(x) \Rightarrow \forall y(R(y) \Rightarrow P(x, y))) \Rightarrow \forall y(R(y) \Rightarrow \forall x(Q(x) \Rightarrow P(x, y)))$. (*)

In this exercise, the formula $P(x, y)$ can be replaced by any other formula. However, $Q(x)$ can only be replaced by a formula that does not have y as a free variable, and $R(y)$ can only be replaced by a formula that does not have x as a free variable. Explain the reason for these constraints.

Exercise 104 (Find the Mistake)
Consider the following formula:

$$\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x\ (P(x) \wedge Q(x)).$$

Among the following three natural deduction proofs, only one is correct. Identify the correct proof, and justify why the other two are incorrect.

1.

context	number	proof	justification
1	1	`Suppose` $\exists x\ P(x) \wedge \forall x\ Q(x)$	
1	2	$\exists x\ P(x)$	$\wedge$E1 1
1	3	$\forall x\ Q(x)$	$\wedge$E2 1
1,4	4	`Suppose` $P(x)$	
1,4	5	$Q(x)$	$\forall$E 3, x
1,4	6	$P(x) \wedge Q(x)$	$\wedge$I 4,5
1,4	7	$\exists x(P(x) \wedge Q(x))$	$\exists$I 6, x
1	8	`Thus` $P(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$I 4,7
1	9	$\exists x(P(x) \wedge Q(x))$	$\exists$E 2,8
	10	`Thus` $\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$I 1,9

2.

context	number	proof	justification
1	1	`Suppose` $\exists x\ P(x) \wedge \forall x\ Q(x)$	
1	2	$\exists x\ P(x)$	$\wedge$E1 1
1	3	$\forall x\ Q(x)$	$\wedge$E2 1
1,4	4	`Suppose` $P(x)$	
1,4	5	$Q(x)$	$\forall$E 3, x
1,4	6	$P(x) \wedge Q(x)$	$\wedge$I 4,5
1	7	`Thus` $P(x) \Rightarrow P(x) \wedge Q(x)$	$\Rightarrow$I 4,6
1	8	$P(x) \wedge Q(x)$	$\exists$E 2,7
1	9	$\exists x(P(x) \wedge Q(x))$	$\exists$I 9, x
	10	`Thus` $\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$I 1,9

3.

context	number	proof	justification
1	1	`Suppose` $\exists x\ P(x) \wedge \forall x\ Q(x)$	
1	2	$\exists x\ P(x)$	$\wedge$E1 1
1	3	$\forall x\ Q(x)$	$\wedge$E2 1
1,4	4	`Suppose` $P(x)$	
1	5	`Thus` $P(x) \Rightarrow P(x)$	$\Rightarrow$I 4,4
1	6	$P(x)$	$\exists$E 2,6
1	7	$Q(x)$	$\forall$E 3, x
1	8	$P(x) \wedge Q(x)$	$\wedge$I 7,8
1	9	$\exists x(P(x) \wedge Q(x))$	$\exists$I 9, x
	10	`Thus` $\exists x\ P(x) \wedge \forall x\ Q(x) \Rightarrow \exists x(P(x) \wedge Q(x))$	$\Rightarrow$I 1,10

Exercise 105 (Copy)
Prove $\forall x \forall y P(x, y) \Rightarrow \forall x \forall y P(y, x)$ using natural deduction with as few copies as possible.

Exercise 106 (Natural Deduction)
Prove the following formulas using natural deduction (note that Q is a propositional variable):

1. $\forall x(Q \wedge P(x)) \Rightarrow Q \wedge \forall x P(x)$.
2. $Q \wedge \forall x P(x) \Rightarrow \forall x(Q \wedge P(x))$.
3. $\forall x(Q \vee P(x)) \Rightarrow Q \vee \forall x P(x)$. (**)
4. $Q \vee \forall x P(x) \Rightarrow \forall x(Q \vee P(x))$.
5. $\exists x(Q \wedge P(x)) \Rightarrow Q \wedge \exists x P(x)$.
6. $Q \wedge \exists x P(x) \Rightarrow \exists x(Q \wedge P(x))$.
7. $\exists x(Q \vee P(x)) \Rightarrow Q \vee \exists x P(x)$.
8. $Q \vee \exists x P(x) \Rightarrow \exists x(Q \vee P(x))$. (*)

In this exercise, the formula $P(x)$ can be replaced by any formula. However, Q can be replaced only by a formula that does not have x as a free variable.

Exercise 107 (Proof)
Prove the formula $\neg\exists x P(x) \Rightarrow \forall x \neg P(x)$. Verify that $P(x)$ can be replaced by any formula. Note: the reciprocal to this formula is proven in example 6.1.6.

Exercise 108 (Equality)
Prove the following formulae:

1. $R(a, c) \wedge (a = b) \Rightarrow R(b, c)$.
2. $x = y \Rightarrow f(x, z) = f(y, z)$.
3. $\forall x \exists y(x = y)$.
4. $\exists x \forall y\ x = y \Rightarrow \forall x \forall y\ x = y$. (*)

Exercise 109 (Equality,)**
Prove that the second definition of " there exists one and only one" (see subsection 6.2.3) implies the first, that is, prove the formula: $\exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$.

Exercise 110 (Induction and natural deduction,*)**
We can define addition using the following formulas:

(a) $\forall n(n + 0 = n)$.

(b) $\forall n \forall p(n + s(p) = s(n + p))$.

These two formula allow us to do additions: we can prove, among other, that $s(0) + s(0) = s(s(0))$. But they do not allow the proof of more general properties of addition, which require the recurrence principle.

1. Show that from hypotheses (a) and (b), we cannot deduce $\forall n(0 + n = n)$.
2. We give the name $P(n)$ to the property above and describe the recurrence principle on this property as follows:

(c) $\forall n(P(n) \Leftrightarrow 0 + n = n)$.

(d) $P(0) \wedge \forall n(P(n) \Rightarrow P(s(n))) \Rightarrow \forall n P(n)$.

Prove using natural deduction that $(a) \wedge (b) \wedge (c) \wedge (d) \Rightarrow \forall n P(n)$.

Exercise 111 (Exam 2009)
All of the following proofs must be justified.

1. Prove using natural deduction that this formula is valid:
   $(\exists x\ p(x) \Rightarrow \forall x\ q(x)) \Rightarrow \forall x(p(x) \Rightarrow q(x))$.

2. Prove using natural deduction that this formula is valid:
   $\exists x\ p(x) \wedge \forall x(p(x) \Rightarrow p(f(x))) \Rightarrow \exists x\ p(f(f(x)))$.

3. We denote by $f^n(x)$ the term obtained applying f n times to x.
   For instance, $f^0(x) = x, f^1(x) = f(x), f^2(x) = f(f(x))$.
   Let Γ, Δ be two formula sets and A, B two formulas. We remind $\Gamma \vdash A$ is true if there is a proof of A in the *environment* Γ.
   We give some trivial properties of the $\vdash$ relation

Monotony : if $\Gamma \vdash A$ and $\Gamma \subset \Delta$ then $\Delta \vdash A$

Composition : if $\Gamma \vdash A$ and $\Gamma \vdash A \Rightarrow B$ then $\Gamma \vdash B$

(a) Prove using natural deduction of the following property:
$\forall x(p(x) \Rightarrow p(f(x))) \vdash \exists x p(f^n(x)) \Rightarrow \exists x\ p(f^{n+1}(x))$

(b) Deduce from the above property, monotony and composition that *for any* natural integer n :
$\exists x\ p(x), \forall x(p(x) \Rightarrow p(f(x))) \vdash \exists x\ p(f^n(x))$.

Exercise 112 (Exam 2012)
Prove the following formulas using first-order natural deduction:

1. $\neg\forall x P(x) \vee \neg\exists y Q(y) \Rightarrow \neg(\forall x P(x) \wedge \exists y Q(y))$

2. $\forall x \forall y(P(y) \Rightarrow R(x)) \Rightarrow \exists y P(y) \Rightarrow \forall x R(x)$

3. $\neg\forall x \neg P(x) \Rightarrow \exists x P(x)$

Exercise 113 (Exam 2013)
Prove the following formulae using first order natural deduction.

1. $\exists x(Q(x) \Rightarrow P(x)) \wedge \forall x Q(x) \Rightarrow \exists x P(x)$

2. $\forall x\ \forall y(R(x, y) \Rightarrow \neg R(y, x)) \Rightarrow \forall x \neg R(x, x)$

Exercise 114 (Questions from various exams)
Prove the following formulae using first order natural deduction.

1. $\exists x(P(x) \vee Q(x)) \wedge \forall x \neg Q(x) \Rightarrow \exists x P(x)$

2. $\forall x(P(x) \Rightarrow Q(x)) \wedge \exists x(P(x) \wedge R(x)) \Rightarrow \exists x(Q(x) \wedge R(x))$

3. $\exists x \neg(P(x) \vee \neg P(x)) \Rightarrow \forall x P(x)$

Partiel INF402

Frédéric Prost Benjamin Wack

16 mars 2023
2 pages
Total : 120 points
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices. L'épreuve sera notée sur 120 points.
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Formalisation (20 points))

Considérons le raisonnement suivant.

- *Dieu est omnibienveillant.*
- *Si la causalité était transitive, on en déduirait que : si Dieu est la cause de l'existence de l'humanité, il s'ensuit que Dieu a causé le changement climatique si c'est l'existence de l'humanité qui l'a causé.*
- *Or, tout bon scientifique doit admettre que c'est l'activité humaine qui a causé les problèmes de climat actuels.*
- *Mais bien que tout bon croyant sache que Dieu a créé l'humanité, un Dieu responsable du changement climatique global ne peut pas être omnibienveillant.*
- *On en déduit que la causalité est non-transitive.*

1. *Formalisez ce raisonnement en logique propositionnelle, en reformulant les affirmations si nécessaire. Ne perdez pas de temps à comprendre les affirmations, concentrez-vous sur les connecteurs logiques.*
2. *Trouvez une assignation qui satisfait les hypothèses.*
3. *Que pouvez-vous en conclure à propos de ce raisonnement ?*

Réponse:

On définit les variables propositionnelles :

P1 :	Dieu est omnibénévolent.
P2 :	Dieu a causé l'existence de l'humanité.
P3 :	Dieu a causé le changement climatique.
Q :	La causalité est transitive.
R :	L'humanité est la cause du changement climatique.

Le raisonnement se traduit par les formules :

$$\begin{array}{l} P1 \\ Q \implies (P2 \implies (R \implies P3)) \\ R \\ P2 \wedge (P3 \implies \neg P1) \\ \models \neg Q \end{array}$$

L'assignation suivante :

P1 :	$\top$
P2 :	$\top$
P3 :	$\bot$
Q :	$\bot$
R :	$\top$.

est le seul modèle des hypothèses.
Comme c'est aussi un modèle de $\neg Q$, le raisonnement est correct.

□

Exercice 2 (Tables de vérité et formes normales (20 points))
Soit $A = \neg x \Rightarrow y \vee z \Rightarrow \neg(z \wedge y) \Rightarrow x$.

1. *Réécrivez cette formule sous forme d'arbre.* *(3 points)*
2. *Construisez la table de vérité de A.* *(5 points)*
3. *Déterminez une forme normale disjonctive de A et simplifiez-la le plus possible.* *(5 points)*
4. *Déterminez une forme normale conjonctive de A et simplifiez-la le plus possible.* *(5 points)*
5. *Donnez un modèle et un contre-modèle de A.* *(2 points)*

Réponse:

1. La formule stricte est $A = \Big(\neg x \Rightarrow \big((y \vee z) \Rightarrow (\neg(z \wedge y) \Rightarrow x)\big)\Big)$.
   qui peut ensuite facilement être écrite sous forme d'arbre.
2. La table de vérité est :

x	y	z	$\neg x$	$y \vee z$	$\neg(y \wedge z)$	A
0	0	0	1	0	1	1
0	0	1	1	1	1	0
0	1	0	1	1	1	0
0	1	1	1	1	0	1
1	0	0	0	0	1	1
1	0	1	0	1	1	1
1	1	0	0	1	1	1
1	1	1	0	1	0	1

3. La FND est $x + \bar{y}.\bar{z} + y.z$
4. La FNC est $(x + y + \bar{z}).(x + \bar{y} + z)$
5. Des modèles et contre-modèles se lisent facilement dans la table de vérité et/ou sur les formes normales.

□

Exercice 3 (Résolution (25 points))

Soient

$$F = \neg(\neg p \wedge \neg q \wedge \neg r) \wedge (\neg p \vee q \vee r)$$

et

$$G = q \vee r$$

Dans tout cet exercice, les seules équivalences remarquables autorisées sont la distributivité et les lois de De Morgan. En particulier, il est interdit d'utiliser les règles de simplification.

1. *Transformez F et G en produit de clauses.*
   Si nécessaire dans la suite de l'exercice, vous pourrez également calculer les clauses pour $\neg F$ et/ou $\neg G$.
2. *Parmi les propriétés suivantes, dites lesquelles sont vérifiées, et justifiez vos réponses en utilisant la résolution et les lois de De Morgan :*
   (a) $F \models G$
   (b) $G \models F$
   (c) $F \equiv G$
   (d) $\neg F \vee G$ *est une tautologie.*
   (e) $\neg F \vee G$ *est une contradiction.*

Réponse: Considérons les clauses correspondant aux formules F et G :

$$F_c = \{(p + q + r), (\overline{p} + q + r)\}$$

$$G_c = \{q + r\}$$

1. $F \models G$ si et seulement si $F \cup \neg G$ est insatisfaisable, ie permet de démontrer l'absurde par résolution.

$$F \cup \neg G = \{(p + q + r), (\overline{p} + q + r), \neg q, \neg r\}$$

1.	$p + q + r$	*Hypothse*
2.	$\overline{p} + q + r$	*Hypothse*
3.	$q + r$	*Resolvant de* 1.2.
4.	$\neg q$	*Hypothse*
5.	r	*Resolvant de* 3.4.
6.	$\neg r$	*Hypothse*
7.	$\perp$	*Resolvant de* 5.6.

2. Comme à la question précédente avec $G \cup \neg F$

$$G_c \cup \neg F_c = \{q + r, p + \overline{q}, p + \overline{r}, \overline{q} + \overline{p}, \overline{q}, \overline{q} + \overline{r}, \overline{p} + \overline{r}, \overline{q} + \overline{r}, \overline{r}\}$$

1.	$q + r$	*Hypothse*
2.	$\overline{r}$	*Hypothse*
3.	q	*Resolvant de* 1.2.
4.	$\overline{q}$	*Hypothse*
5.	$\perp$	*Resolvant de* 3.4.

3. $F \equiv G$ puisque $F \models G$ et $G \models F$.
4. $\neg F \vee G$ est une tautologie puisque $F \equiv G$ (loi du tiers exclus).
5. $\neg F \vee G$ n'est pas une contradiction puisque c'est une tautologie.

□

Exercice 4 (DPLL (20 points))

Déterminez **en appliquant l'algorithme DPLL** *si les ensembles de clauses suivants sont satisfaisables. Le cas échéant, vous donnerez le modèle exhibé par DPLL.*
Vous donnerez une trace complète du déroulement de l'algorithme étiquetée par les règles utilisées.

1. $\{a+b+\overline{c},\ \overline{a}+c+d,\ e+a+b,\ \overline{e}+c+e,\ b+\overline{c},\ \overline{b}+d,\ \overline{a}+\overline{c},\ c+\overline{d}\}$ *(10 points)*
2. $\{a+d,\ a+\overline{b}+\overline{c}+\overline{d},\ a+b+c+\overline{d},\ \overline{a}+\overline{b}+\overline{c}+d,\ \overline{a}+c,\ \overline{b}+a+c,\ \overline{c}+b,\ \overline{d}+\overline{a}\}$ *(10 points)*

Réponse:

1. $\{a+b+\overline{c},\ \overline{a}+c+d,\ e+a+b,\ \overline{e}+c+e,\ b+\overline{c},\ \overline{b}+d,\ \overline{a}+\overline{c},\ c+\overline{d}\}$
   RED : $\{\overline{a}+c+d,\ e+a+b,\ b+\overline{c},\ \overline{b}+d,\ \overline{a}+\overline{c},\ c+\overline{d}\}$
   ELI $e=1$: $\{\overline{a}+c+d,\ b+\overline{c},\ \overline{b}+d,\ \overline{a}+\overline{c},\ c+\overline{d}\}$
   ELI $a=0$: $\{b+\overline{c},\ \overline{b}+d,\ c+\overline{d}\}$
   Branche $b=0$: $\{\overline{c},\ c+\overline{d}\}$
   ELI $d=0$: $\{\overline{c}\}$
   ELI $c=0$: $\{\}$
   On a un modèle $a=0, b=0, c=0, d=0, e=1$
2. $\{a+d,\ a+\overline{b}+\overline{c}+\overline{d},\ a+b+c+\overline{d},\ \overline{a}+\overline{b}+\overline{c}+d,\ \overline{a}+c,\ \overline{b}+a+c,\ \overline{c}+b,\ \overline{d}+\overline{a}\}$
   $\{a+d,\ a+\overline{b}+\overline{c}+\overline{d},\ a+b+c+\overline{d},\ \overline{a}+\overline{b}+\overline{c}+d,\ \overline{a}+c,\ \overline{b}+a+c,\ \overline{c}+b,\ \overline{d}+\overline{a}\}$
   — Branche $a=0$: $\{d,\ \overline{b}+\overline{c}+\overline{d},\ b+c+\overline{d},\ \overline{b}+c,\ \overline{c}+b\}$
     RU $d=1$: $\{\overline{b}+\overline{c},\ b+c,\ \overline{b}+c,\ \overline{c}+b\}$
     — Branche $b=0$: $\{c,\ \overline{c}\}$
       RU : $\perp$
     — Branche $b=1$: $\{\overline{c},\ c\}$
       RU : $\perp$
   — Branche $a=1$: $\{\overline{b}+\overline{c}+d,\ c,\ \overline{c}+b,\ \overline{d}\}$
     RU $c=1, d=0$: $\{\overline{b},\ b\}$
     RU : $\perp$
   L'ensemble de clauses est insatisfaisable.

□

Exercice 5 (Déduction naturelle (20 points))
Prouver, **par déduction naturelle**, *les formules suivantes.*

Chaque formule de la preuve devra être justifiée ; il vous est recommandé de présenter vos preuves sous forme de tableaux comme vu en cours.

1. $A \ = \ (p \vee q \Rightarrow s \wedge t) \Rightarrow (q \Rightarrow s)$ *(10 points)*
2. $B \ = \ (p \vee \neg q) \Rightarrow \big((p \Rightarrow \neg q) \Rightarrow (q \Rightarrow r)\big)$ *(10 points)*

Réponse:

1. $A \ = \ (p \vee q \Rightarrow s \wedge t) \Rightarrow (q \Rightarrow s)$

```
1   assume p + q => s & t.
2     assume q.
3       p + q.                                  +I2 2
4       s & t.                                  =>E 1,3
5       s.                                      &E1 4
6     therefore q => s.                         =>I 2,5
7   therefore (p + q => s & t) => q => s.       =>I 1,6
```

2. $B \ = \ (p \vee \neg q) \Rightarrow \big((p \Rightarrow \neg q) \Rightarrow (q \Rightarrow r)\big)$

```
1   assume p + -q.
2     assume p => -q.
3       assume q.
4         assume p.
5           -q.                                 =>E 2,4
6           F.                                  =>E 5,3
7           r.                                  Efq 6
8         therefore p => r.                     =>I 4,7
9         assume -q.
```

```
10          F.                                          =>E 9,3
11          r.                                          Efq 10
12        therefore -q => r.                            =>I 9,11
13        r.                                            +ES 1,8,12
14      therefore q => r.                               =>I 3,13
15    therefore (p => -q) => q => r.                    =>I 2,14
16  therefore p + -q => (p => -q) => q => r.            =>I 1,15
```

□

Exercice 6 (Démonstration par récurrence, exo du poly (15 points))

Démontrez que toute formule construite avec une seule variable p, uniquement les connecteurs binaires $\vee$ et $\wedge$ et sans négation est équivalente à une formule de taille 0.

Si vous trouvez des cas très semblables, vous pouvez raccourcir la démonstration en admettant une partie d'entre eux.

Réponse: Nous effectuons une preuve par récurrence sur la taille des formules. Considérons une formule F construite avec une seule variable p, uniquement $\vee$ et $\wedge$ et sans négation.

Cas de base : $|F| = 0$. La récurrence est trivialement vérifiée dans ce cas.

Induction : $|F| = n + 1$ et supposons que toutes les formules construite avec une seule variable p, uniquement $\vee$ et $\wedge$ et sans négation de taille inférieure ou égale à n sont équivalentes à une formule de taille 0. Nous avons deux cas duaux, où $|A| \leq n$ et $|B| \leq n$, donc par hypothèse de récurrence A et B sont donc équivalentes à une formule de taille 0, c'est-à-dire 0, 1, ou p :

- $F = (A \vee B)$. Nous avons donc 9 cas à examiner :
  - $A = 0$
    - $B = 0$, donc $F = 0$.
    - $B = 1$, donc $F = 1$.
    - $B = p$, donc $F = p$.
  - $A = 1$
    - $B = 0$, donc $F = 1$.
    - $B = 1$, donc $F = 1$.
    - $B = p$, donc $F = 1$.
  - $A = p$
    - $B = 0$, donc $F = p$.
    - $B = 1$, donc $F = 1$.
    - $B = p$, donc $F = p$.
- $F = (A \wedge B)$. Nous avons donc 9 cas à examiner :
  - $A = 0$
    - $B = 0$, donc $F = 0$.
    - $B = 1$, donc $F = 0$.
    - $B = p$, donc $F = 0$.
  - $A = 1$
    - $B = 0$, donc $F = 0$.
    - $B = 1$, donc $F = 1$.
    - $B = p$, donc $F = p$.
  - $A = p$
    - $B = 0$, donc $F = 0$.
    - $B = 1$, donc $F = p$.
    - $B = p$, donc $F = p$.

□

Examen INF402

Frédéric Prost Benjamin Wack

16 mai 2023
2 pages
Total : 120 points (noté sur 100)
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices. **L'épreuve sera notée sur 100 points (total divisé par 5), il est donc possible de faire l'impasse sur un des exercices sans pénalité.**
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Formalisation (20 points))**.**
Nous considérons la signature $\Sigma = \{animal^{r1},\ plante^{r1},\ mange^{r2},\ pluspetit^{r2}\}$.

1. (12 points) *Exprimer en logique du premier ordre et en utilisant la signature* Σ *les phrases suivantes :*
   (a) *Il existe des animaux qui mangent à la fois des plantes et des animaux.*
   (b) *Aucune plante ne mange d'animal.*
   (c) *Tous les animaux mangent soit toutes les plantes, soit certains animaux plus petits qu'eux.*
   (d) *Les animaux qui mangent d'autres animaux mangent tous les animaux plus petits qu'eux.*
2. (8 points) *Exprimer en français les propositions logiques suivantes :*
   (a) $\forall x \neg(animal(x) \Leftrightarrow plante(x))$
   (b) $\exists x(plante(x) \wedge \forall y(plante(y) \wedge x \neq y \Rightarrow pluspetit(y,x))$
   (c) $\exists x \exists y(animal(x) \wedge animal(y) \wedge mange(x,y) \wedge \forall z(mange(x,z) \Rightarrow y = z))$

□

Exercice 2 (Expansions (20 points))**.**
À l'aide de la méthode des expansions, déterminer ***un contre-modèle*** *de chacune de ces formules :*

1. (10 points) $A = \big(\forall x Q(x) \implies \forall x P(x)\big) \implies \forall x\big(Q(x) \implies P(x)\big)$.
2. (10 points) $B = \forall x \exists y\big((x = y) \implies P(x,y)\big) \implies \forall x \forall y\big(P(x,y) \implies (x = y)\big)$.

□

Exercice 3 (Unification (15 points))**.**
Pour chacune des équations suivantes, déterminez si elle admet une solution, et si oui déterminez un unificateur le plus général. Vous utiliserez pour cela l'algorithme du cours dont vous détaillerez les étapes en donnant à chaque fois le nom des règles utilisées.
a et b sont des constantes ; f, g et h sont des fonctions.

1. (5 points) $f\big(x, f(u,x)\big) = f\big(f(y,a), f(z, f(b,z))\big)$
2. (5 points) $h\big(z, f(y, g(a,x))\big) = h\big(f(b,b), f(z, g(x, h(x,x)))\big)$
3. (5 points) $f\big(h(g(a,x), g(z,z)), g(z,z)\big) = f\big(h(y,y), y\big)$

□

Exercice 4 (Skolémisation (15 points)).

Soit A la formule suivante : $A = \Big(\exists x \forall y Q(x,y) \wedge \forall x\big(Q(x,x) \Rightarrow \exists y R(y,x)\big)\Big) \Rightarrow \exists y \exists x R(x,y)$

Montrez que A est valide en montrant que $\neg A$ est contradictoire par la méthode de Herbrand, en suivant les étapes suivantes :

1. (10 points) *Skolémisez $\neg A$ puis mettez-la en forme clausale.*
2. (5 points) *Proposez des instances contradictoires des clauses obtenues et montrez la contradiction par résolution propositionnelle.*

□

Exercice 5 (Résolution (10 points)).

Considérons l'ensemble suivant de clauses du premier ordre :

$$\Gamma = \{\overline{P(x)} + \overline{P(f(a))} + Q(y), P(y), \overline{P(g(b,x))} + \overline{Q(b)}, \overline{P(f(x))} + Q(x)\}$$

Nous voulons montrer que cet ensemble de clauses est insatisfaisable par instanciation ou par résolution. ***Au choix :*** *(vous n'obtiendrez pas le double de points en utilisant les deux méthodes)*

1. (10 points) *Trouvez des instances contradictoires de ces clauses et montrez par résolution propositionnelle que ces instances sont contradictoires.*

   OU

2. (10 points) *Donnez une preuve directe du caractère contradictoire de Γ par factorisation, copie et résolution binaire. Il est possible que toutes les règles ne soient pas utilisées.*

□

Exercice 6 (Déduction naturelle (20 points)).

Démontrez les formules suivantes en déduction naturelle au premier ordre. Comme vu en cours et en TD, on numérotera les lignes de la preuve, et on indiquera pour chaque ligne la formule démontrée, le contexte, la règle appliquée et les prémisses utilisées.

1. $\forall y \exists x \neg Q(x,y) \Rightarrow \neg \forall x \forall y Q(x,y)$
2. $\forall x \forall y (P(x) \wedge \neg P(y) \Rightarrow x \neq y)$

□

Exercice 7 (Ensemble complet et incomplet de connecteurs, exercice du poly (20 points)).

Un ensemble de constantes et de connecteurs est dit complet*, si toute fonction booléenne est exprimable avec ces constantes et ces connecteurs. Nous avons notamment vu en cours un théorème qui exprime que l'ensemble $\{\bot, \top, \neg, \vee, \wedge\}$ est complet.*

1. *Soit $\mid$ l'opération suivante : $x \mid y$ est vrai si et seulement si ni l'un, ni l'autre ne sont vrais, autrement dit $x \mid y = 1$ si et seulement si $x = 0$ et $y = 0$. Cette opération est aussi appelée* ni *car $x \mid y$ est vrai si et seulement si ni x, ni y ne sont vrais.*

   Montrer que cette opération (seule !) est complète.

2. *Montrer que l'ensemble $\{\top, \Rightarrow\}$ est incomplet.*

   Indication *: on pourra montrer que toute formule construite uniquement avec $\top$ et $\Rightarrow$ prend la valeur 1 quand toutes ses variables prennent la valeur 1. On en déduira alors assez facilement une fonction booléenne qui ne peut pas être exprimée avec cet ensemble.*

□

Corrigés

Exercice 1, page 1

1. $\exists x \exists y \exists z (animal(x) \wedge plante(y) \wedge animal(z) \wedge mange(x,y) \wedge mange(x,z))$
2. $\forall x (plante(x) \Rightarrow \forall y (animal(y) \Rightarrow \neg mange(x,y)))$
3. $\forall x (animal(x) \Rightarrow \forall y (plante(y) \Rightarrow mange(x,y)) \vee \exists y (animal(y) \wedge pluspetit(y,x) \wedge mange(x,y)))$
4. $\forall x (animal(x) \wedge \exists y (animal(y) \wedge mange(x,y)) \Rightarrow \forall z (animal(z) \wedge pluspetit(z,x) \Rightarrow mange(x,z))$
5. Tout être est soit une plante soit un animal, mais jamais les deux à la fois.
6. Il existe une plante plus grande que toutes les autres.
7. Il existe un animal qui ne mange qu'une seule espèce d'animal.

Exercice 2, page 1

1. La 1-expansion est valide : $(Q(0) \implies P(0)) \implies (Q(0) \implies P(0))$
   La 2-expansion est : $(Q(0).Q(1) \implies P(0).P(1)) \implies (Q(0) \implies P(0)).(Q(1) \implies P(1))$ Un contre modèle est $Q_I = \{0\}$ et $P_I = \{1\}$ ou $\emptyset$ (on a un contre modèle symétrique avec Q_I=$\{1\}$).
2. La 1-expansion est valide : $((0 = 0) \implies P(0,0)) \implies (P(0,0) \implies 0 = 0)$ pour toutes les interprétations de P.
   La 2-expansion est :
   $(((0 = 0) \implies P(0,0)) + ((0 = 1) \implies P(0,1)))$.
   $(((1 = 0) \implies P(1,0)) + ((1 = 1) \implies P(1,1))) \implies (P(0,0) \implies 0 = 0).(P(1,0) \implies 1 = 0)$.
   $(P(0,1) \implies 0 = 1).(P(1,1) \implies 1 = 1)$
   Un contre modèle est donnée par l'interprétation $P_I = \{(0,1),(1,0)\}$ (ou contenant un de ces deux couples).

Exercice 3, page 1

1. Utilisation de décomposition : $(1) x = f(y,a)$, $(2) u = z$, $(3) x = f(b,z)$.
   Elimination de (1) dans (3) donne (4) $f(y,a) = f(b,z)$.
   Utilisation de décomposition sur (4) donne (5) $y = b$, (6) $z = a$,
   on élimine z dans (2) en utilisant (6).
   On trouve $\sigma = \{x := f(b,a), y := b, u := a, z := a\}$.
2. Utilisation de décomposition : (1) $z = f(b,b)$, $(2) f(y, g(a,x)) = f(z, g(x, h(x,x)))$.
   Décomposition de (2) donne $(3) y = z$, (4) $g(a,x) = g(x, h(x,x))$.
   Décomposition de (4) donne $(5) x = a$, (6) $x = h(x,x)$.
   Échec d'élimination en (6) l'algorithme s'arrête pas d'unification possible.
3. Utilisation de décomposition : $(1) h(g(a,x), g(z,z)) = h(y,y)$, $(2) y = g(z,z)$.
   Décomposition de (1) donne $(3) g(a,x) = y$ et $(4) y = g(z,z)$.
   On peut supprimer (4) qui est la même équation que (2).
   On élimine y dans (3) avec (2) ce qui donne $(5) g(z,z) = g(a,x)$.
   Décomposition de (5) donne (6) $z = a$ et (7) $z = x$.
   On trouve $\sigma = \{z := a, x := a, y = g(a,a)\}$.

Exercice 4, page 2

1. — $\neg A \equiv \exists x \forall y Q(x,y) \wedge \forall x\big(\neg Q(x,x) \vee \exists y R(y,x)\big) \wedge \neg \exists y \exists x R(x,y)$
   — Forme normale : $\exists x \forall y Q(x,y) \wedge \forall x\big(\neg Q(x,x) \vee \exists y R(y,x)\big) \wedge \forall y \forall x \neg R(x,y)$
   — Forme propre : $\exists x \forall y Q(x,y) \wedge \forall z\big(\neg Q(z,z) \vee \exists t R(t,z)\big) \wedge \forall u \forall v \neg R(v,u)$
   — Élimination des $\exists$: $\forall y Q(a,y) \wedge \forall z\big(\neg Q(z,z) \vee R(f(z),z)\big) \wedge \forall u \forall v \neg R(v,u)$
   — Élimination des $\forall$: $\forall y \forall z \forall u \forall v\Big(Q(a,y) \wedge \big(\neg Q(z,z) \vee R(f(z),z)\big) \wedge \neg R(v,u)\Big)$
   — Forme clausale : $Q(a,y),\ \overline{Q(z,z)} + R(f(z),z),\ \overline{R(v,u)}$
2. On instancie $y := a, z := a, v := f(a), u := a$.
   Alors $Q(a,a)$ et $\overline{Q(a,a)} + R(f(a),a)$ ont pour résolvant $R(f(a),a)$ qui est contradictoire avec $\overline{R(f(a),a)}$.

Exercice 5, page 2

— On effectue $\{x := f(a), y := b\}$ dans $\overline{P(x)} + \overline{P(f(a))} + Q(y)$ ce qui donne $\overline{P(f(a))} + Q(b)$ on applique la résolution avec $P(y)$ instanciée avec $\{y := f(a)\}$ ce qui donne la résolvante (1) $Q(b)$.
On fait une autre instance de $P(y)$ avec $\{y := g(b,a)\}$ ce qui donne $P(g(b,a))$. En instantiant $\overline{P(g(b,x)), \overline{Q(b)}}$ avec $\{x := a\}$ on obtient $\overline{P(g(b,a)), \overline{Q(b)}}$. En appliquant la résolution à ces deux instantiations on obtient (2) $\overline{Q(b)}$.
On conclut en faisant une résolution sur (1) et (2) qui donne la clause vide.

— Par factorisation de $\overline{P(x)} + \overline{P(f(a))} + Q(y)$ on obtient $\overline{P(f(a))} + Q(y)$. Par résolution propositionnelle avec $P(y)$ on obtient $Q(y)$. Par résolution propositionnelle entre $P(y)$ et $\overline{P(g(b,x)) + \overline{Q(b)}}$ on obtient $Q(b)$ avec une dernière résolution propositionnelle entre $Q(y)$ et $\overline{Q(b)}$ on trouve la clause vide.

Exercice 6, page 2

1. *(10 points)* $\forall y \exists x \neg Q(x,y) \Rightarrow \neg \forall x \forall y Q(x,y)$

contexte	numéro	preuve	justification
1	1	`Supposons` $\forall y \exists x \neg Q(x,y)$	
1,2	2	`Supposons` $\forall x \forall y Q(x,y)$	
1,2	3	$\exists x \neg Q(x,y)$	$\forall E y$ 2
1,2,4	4	`Supposons` $\neg Q(x,y)$	
1,2,4	5	$\forall y Q(x,y)$	$\forall E x$ 4
1,2,4	6	$Q(x,y)$	$\forall E y$ 5
1,2,4	7	$\perp$	$\Rightarrow E$ 4,6
1,2	8	`Donc` $\neg Q(x,y) \Rightarrow \perp$	$\Rightarrow I$ 4, 7
1,2	9	$\perp$	$\exists E$ 3,8
	10	`Donc` $\neg \forall x \forall y Q(x,y)$	$\Rightarrow I$ 2, 9
	11	`Donc` $\forall y \exists x \neg Q(x,y) \Rightarrow \neg \forall x \forall y Q(x,y)$	$\Rightarrow I$ 1, 10

2. *(10 points)* $\forall x \forall y (P(x) \wedge \neg P(y) \Rightarrow x \neq y)$

contexte	numéro	preuve	justification
1	1	`Supposons` $P(x) \wedge \neg P(y)$	
1,2	2	`Supposons` $x = y$	
1,2	3	$P(x)$	$\wedge E1$ 1
1,2	4	$\neg P(y)$	$\wedge E2$ 1
1,2	5	$P(y)$	*cong* 2,3
1,2	6	$\perp$	$\Rightarrow E$ 4,5
1	7	`Donc` $x \neq y$	$\Rightarrow I$ 2,6
	8	`Donc` $P(x) \wedge \neg P(y) \Rightarrow x \neq y$	$\Rightarrow I$ 1,7
	9	$\forall y (P(x) \wedge \neg P(y) \Rightarrow x \neq y)$	$\forall I$ 8
	10	$\forall x \forall y (P(x) \wedge \neg P(y) \Rightarrow x \neq y)$	$\forall I$ 9

Exercice 7, page 2

1. Montrons que $|$, définie par $x \mid y = 1$ si et seulement si $x = 0$ et $y = 0$, est complète.
   Les connecteurs $\{\neg, \bot, \vee, \wedge\}$ sont définissables à l'aide de $|$:
   — $\neg x = x \mid x$
   — $\bot = \neg x \mid x = (x \mid x) \mid x$
   — $x \vee y = \overline{x \mid y} = (x \mid y) \mid (x \mid y)$
   — $x \wedge y = \bar{x} \mid \bar{y} = (x \mid x) \mid (y \mid y)$
   — enfin on exprime $\top$ comme $\neg\bot = ((x \mid x) \mid x) \mid ((x \mid x) \mid x)$
2. Montrons que l'ensemble $\{1, \Rightarrow\}$ est incomplet.
   Appelons stable pour 1, une fonction booléenne qui prend la valeur 1 quand tous ses arguments ont la valeur 1.
   Il est facile de voir que toute fonction booléenne définie avec les seules opérations $\{1, \Rightarrow\}$ est stable pour 1, car $1 \Rightarrow 1 \equiv 1$ (on peut le faire proprement par récurrence sur la taille de la formule).
   Or $\neg$ n'est pas stable pour 1, donc la fonction $f(x) = \neg x$ n'est pas exprimable.
   Par suite cet ensemble est incomplet.

Partiel INF402

Eskandar Kouicem Benjamin Wack

14 mars 2022
2 pages
Total : 120 points + 5 points de bonus
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices. L'épreuve sera notée sur 120 points + 5 points de bonus.
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Modèles et formes normales (25 points))
Soit $A = a \vee \neg c \Leftrightarrow d \Rightarrow b \wedge d \vee \neg a \Rightarrow b$.

1. *Réécrivez cette formule sous forme d'arbre.* *(3 points)*
2. *Déterminer une forme normale disjonctive de A et la simplifier le plus possible.* *(9 points)*
3. *Déterminer une forme normale conjonctive de A et la simplifier le plus possible.* *(9 points)*
4. *Si c'est possible, donner un modèle de A et justifier la réponse* sans utiliser de table de vérité. *(2 points)*
5. *Si c'est possible, donner un contre-modèle de A et justifier la réponse* sans utiliser de table de vérité. *(2 points)*

Réponse:
- Je donne A sous sa forme stricte : $(a \vee \neg c) \Leftrightarrow (d \Rightarrow (((b \wedge d) \vee \neg a) \Rightarrow b))$.
- En forme normale disjonctive : $a + c.d.\bar{b} + \bar{c}.\bar{d} + \bar{c}.b$
  Ce qui donne 4 modèles : (1) $a = 0$, (2) $b = 0, c = d = 1$, (3) $c = d = 0$, (4) $b = 1, c = 0$.
- En forme normale conjonctive : $(a + \bar{c} + d).(a + \bar{c} + \bar{b}).(a + c + \bar{d} + b)$
  Ce qui donne 3 contre-modèles : (1) $a = d = 0, c = 1$, (2) $a = 0, b = c = 0$, (3) $a = b = c = 0, d = 1$.

□

Exercice 2 (Formalisation (Exo du poly, 15 points))
Formaliser les énoncés suivants en logique propositionnelle :

(a) *Si Pierre est rentré chez lui, alors Jean est allé au cinéma.*

(b) *Marie est à la bibliothèque ou Pierre est rentré chez lui.*

(c) *Si Jean est allé au cinéma, alors Marie est à la bibliothèque ou Pierre est rentré chez lui.*

(d) *Marie n'est pas à la bibliothèque et Jean est allé au cinéma.*

(e) *Pierre est rentré chez lui.*

À l'aide d'une des méthodes vues en cours (au choix), déterminer si le dernier énoncé (e) est conséquence des énoncés (a) (b) (c) et (d).
Réponse: Les faits, Pierre est rentré chez lui, Jean est allé au cinéma, Marie est à la bibliothèque sont dénotés respectivement par les variables p, j, m. Les hypothèses sont formalisées par les formules :

1. $p \Rightarrow j$.
2. $m \vee p$.
3. $j \Rightarrow m \vee p$.

4. $\neg m \wedge j$.

Et la conclusion par p.

En utilisant les équivalences remarquables, nous prouvons que $(p \Rightarrow j) \wedge (m \vee p) \wedge (j \Rightarrow m \vee p) \wedge (\neg m \wedge j) \wedge \neg p$ est contradictoire. Car en enlevant les implications nous avons :

$$(\neg p \vee j) \wedge (m \vee p) \wedge (\neg j \vee m \vee p) \wedge \neg m \wedge j \wedge \neg p.$$

En utilisant $(x \vee \neg y) \wedge y = x \wedge y$, entre la seconde et la quatrième clause nous obtenons :

$$(\neg p \vee j) \wedge p \wedge (\neg j \vee m \vee p) \wedge \neg m \wedge j \wedge \neg p.$$

Nous obtenons une contradiction entre p et $\neg p$. Et donc le raisonnement est correct.

□

Exercice 3 (Résolution (20 points))

Montrez **par résolution** *que les ensembles de formules suivants sont insatisfaisables :*

1. $\{a + \bar{c}, c, \bar{a} + b + \bar{d} + e, b + d, e + \bar{b}, f + g, \bar{g} + f, \bar{f} + \bar{e}\}$ *(10 points)*
2. $\{\bar{q} + m, \bar{q} + \bar{m} + \bar{r}, \bar{q} + \bar{m} + r, \bar{p} + r, \bar{p} + \bar{r}, p + q\}$ *(10 points)*

Réponse:

1	$a + \bar{c}$	Hyp
2	c	Hyp
3	a	Résolvant de 1, 2
4	$\bar{a} + b + \bar{d} + e$	Hyp
5	$b + \bar{d} + e$	Résolvant de 3, 4
6	$b + d$	Hyp
7	$b + e$	Résolvant de 5, 6
8	$e + \bar{b}$	Hyp
9	e	Résolvant 7,8
10	$f + g$	Hyp
11	$\bar{g} + f$	Hyp
12	f	Résolvant de 10,11
13	$\bar{f} + \bar{e}$	Hyp
14	$\bar{e}$	Résolvant de 12,13
15	$\perp$	Résolvant de 9,14

1	$\bar{q} + \bar{m} + \bar{r}$	Hyp
2	$\bar{q} + \bar{m} + r$	Hyp
3	$\bar{q} + \bar{m}$	Résolvant de 1, 2
4	$\bar{q} + m$	Hyp
5	$\bar{q}$	Résolvant de 3, 4
6	$p + q$	Hyp
7	p	Résolvant de 5, 6
8	$\bar{p} + r$	Hyp
9	$\bar{p} + \bar{r}$	Hyp
10	$\bar{p}$	Résolvant de 8,9
11	$\perp$	Résolvant de 7,10

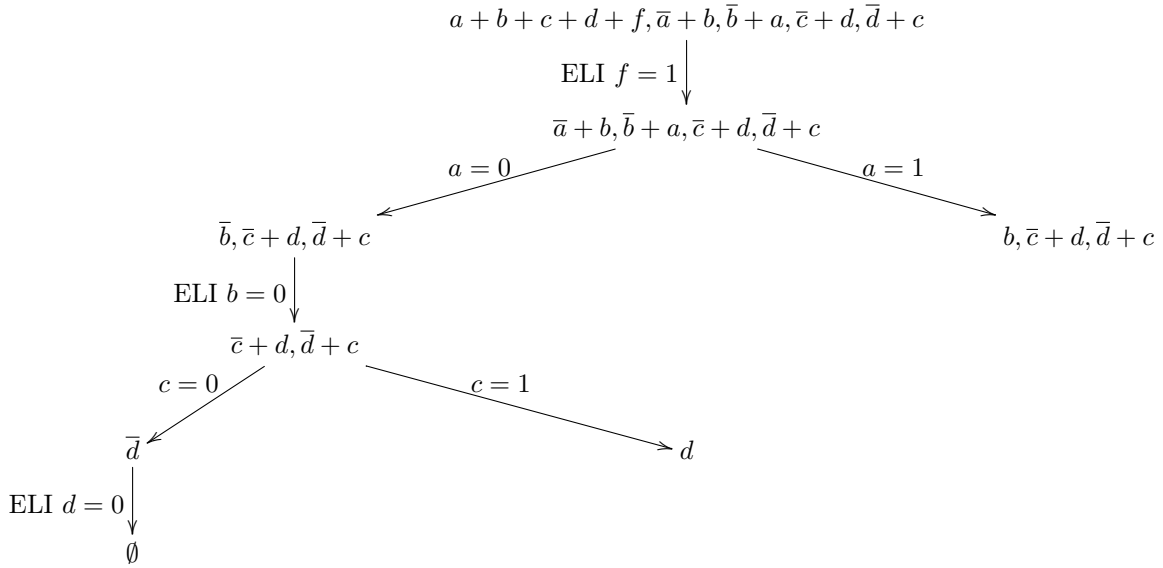
□

Exercice 4 (DPLL (20 points))
Déterminez **en appliquant l'algorithme DPLL** *si les ensembles de clauses suivants sont satisfaisables. Le cas échéant, vous donnerez le modèle exhibé par DPLL.*
Vous donnerez une trace complète du déroulement de l'algorithme étiquetée par les règles utilisées.

1. $\{a+b+c+d+f, \overline{a}+b, \overline{b}+a, \overline{c}+d, \overline{d}+c\}$. *(10 points)*
2. $\{p+q+\bar{r},\ r+q+\bar{p},\ p+r+q,\ r+q,\ q+p,\ \bar{p}+\bar{q},\ \bar{q}+p,\ q+s,\ \bar{s}+\bar{p}\}$ *(10 points)*

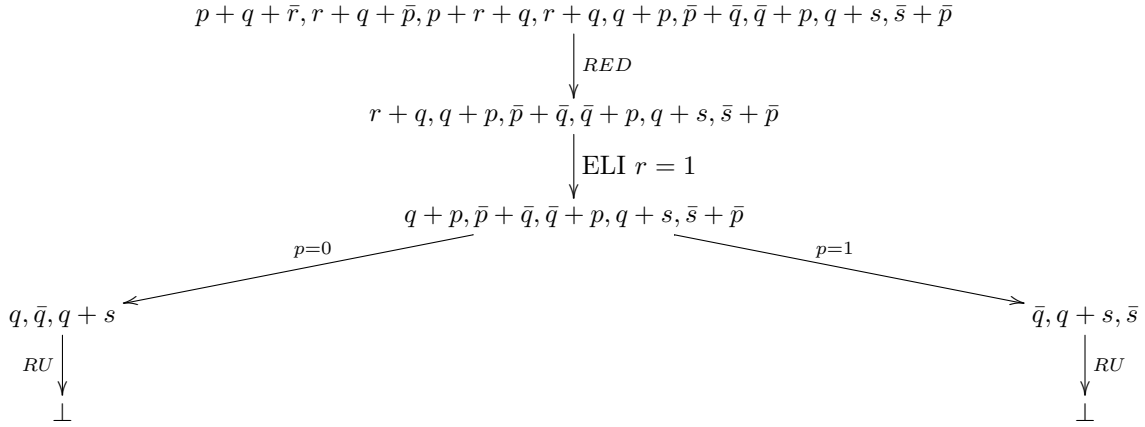
Réponse:

1.



D'après la trace ci-dessus, l'ensemble est satisfaisable. Nous observerons que, d'après le sens de « ou alors », l'algorithme se termine sur la première feuille, où la valeur vrai est retourné. Un modèle est donc $d=0$, $c=1$, $b=0$, $a=0$, et $f=1$.

2.



□

Exercice 5 (Déduction naturelle (25 points))
Prouver, **par déduction naturelle**, *les formules suivantes.*

Chaque formule de la preuve devra être justifiée ; il vous est recommandé de présenter vos preuves sous forme de tableaux comme vu en cours.

1. $((p \vee q) \Rightarrow r) \Rightarrow (p \Rightarrow r)$ *(5 points)*
2. $(p \Rightarrow \neg p) \wedge (\neg p \Rightarrow p) \Rightarrow \perp$ *(10 points)*
3. $p \Rightarrow (\neg p \vee q) \Rightarrow q$ *(10 points)*

Réponse:

1. $((p \vee q) \Rightarrow r) \Rightarrow (p \Rightarrow r)$

```
1    assume (p + q) => r.
2      assume p.
3        p + q.                                          +I1 2
4        r.                                              =>E 1,3
5      therefore p => r.                                 =>I 2,4
6    therefore ((p + q) => r) => (p => r).               =>I 1,5
```

2. $(p \Rightarrow \neg p) \wedge (\neg p \Rightarrow p) \Rightarrow \bot$

```
1    assume (p => -p) & (-p => p).
2      p => -p.                                          &E1 1
3      -p => p.                                          &E2 1
4      assume p.
5        -p.                                             =>E 2,4
6        F.                                              =>E 5,4
7      therefore -p.                                     =>I 4,6
8      p.                                                =>E 3,7
9      F.                                                =>E 7,8
10   therefore (p => -p) & (-p => p) => F.               =>I 1,9
```

3. $p \Rightarrow (\neg p \vee q) \Rightarrow q$

```
1    assume p.
2      assume -p + q.
3        assume -p.
4          F.                                            =>E 3,1
5          q.                                            Efq 4
6        therefore -p => q.                              =>I 3,5
7        assume q.
8        therefore q => q.                               =>I 7,7
9        q.                                              +E 2,6,8
10     therefore -p + q => q.                            =>I 2,9
11   therefore p => -p + q => q.                         =>I 1,10
```

□

Exercice 6 (Démonstration par récurrence (20 points))
Démontrer que toute formule stricte construite uniquement avec :

- *la constante* $\top$,
- *la variable propositionnelle* x,
- *les connecteurs* $\Rightarrow$ *et* $\vee$,
- *et les parenthèses*

est équivalente soit à x *soit à* $\top$.

Réponse: Par récurrence sur la taille de la formule.

Une formule de taille 0 est nécessairement une variable ou une constante, donc en l'occurrence x ou $\top$.

Soit un entier $n > 0$, supposons la propriété vraie pour toute formule de taille strictement inférieure à n. Soit alors A une formule de taille n de la forme décrite dans l'énoncé. A est forcément de la forme $B \Rightarrow C$ ou bien $B \vee C$, où B et C sont des formules de taille strictement inférieure à n, toutes deux construites comme A.

Par hypothèse de récurrence, B et C sont toutes deux équivalentes à x ou $\top$, on est donc ramené à étudier 8 cas :

- $A \equiv x \Rightarrow x \equiv \top$
- $A \equiv x \Rightarrow \top \equiv \top$
- $A \equiv \top \Rightarrow x \equiv x$
- $A \equiv \top \Rightarrow \top \equiv \top$
- $A \equiv x \vee x \equiv x$
- $A \equiv \top \vee x \equiv \top$ et de même pour $x \vee \top$ et $\top \vee \top$

□

Examen INF402

Eskandar Kouicem Benjamin Wack

23 mai 2022
2 pages
Total : 120 points
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices. L'épreuve sera notée sur 120 points.
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Expansions (20 points))**.**
À l'aide de la méthode des expansions, déterminer ***un contre-modèle*** *de chacune de ces formules :*

1. (10 points) $A = \exists x\big(P(x) \Rightarrow \forall y(P(y) \Rightarrow x = y)\big)$.
2. (10 points) $B = \forall x \exists y R(x, y) \Rightarrow \exists y \forall x R(x, y)$.

□

Exercice 2 (Formalisation (20 points, exercice du poly))**.**
Nous considérons $\Sigma = \{f^{r2}, o^{r2}, c^{r2}, j^{r2}, r^{f0}, p^{f1}\}$ *la signature ayant la sémantique donnée ci-dessous.*
— $f(x, y)$: *x est un frère de y.*
— $o(x, y)$: *x est un oncle de y.*
— $c(x, y)$: *x est un cousin de y.*
— $j(x, y)$: *x est plus jeune que y.*
— *r est le diminutif de Robert.*
— $p(x)$ *est le père de x.*

1. (12 points) *Exprimer en logique du premier ordre et en utilisant la signature Σ les phrases suivantes :*
   (a) *Tout frère du père de Robert est un oncle de Robert.*
   (b) *Si les pères de deux enfants sont des frères alors ces deux enfants sont des cousins.*
   (c) *Robert a un cousin plus jeune qu'un des frères de Robert.*
2. (8 points) *Exprimer en français les propositions logiques suivantes :*
   (a) $\neg\exists x\ j(p(x), x)$
   (b) $\forall x \forall y(p(p(x)) = p(p(y)) \Rightarrow c(x, y))$
   (c) $\forall x \exists y(f(x, y) \wedge j(x, y))$
   (d) $\exists x \exists y(f(x, y) \wedge \neg(p(x) = p(y)))$

□

Exercice 3 (Unification (15 points))**.**
Pour chacune des équations suivantes, déterminez si elle admet une solution, et si oui déterminez un unificateur le plus général. Vous utiliserez pour cela l'algorithme du cours dont vous détaillerez les étapes en donnant à chaque fois le nom des règles utilisées.
a et b sont des constantes ; f, g et h sont des fonctions.

1. (5 points) $f(x, g(a), x) = f(y, g(y), b)$
2. (5 points) $f(f(x, y, a), g(y), h(y, b)) = f(z, x, h(b, b))$
3. (5 points) $h(g(x), h(y, x)) = h(g(y), h(g(z), z))$

□

Exercice 4 (Skolémisation (15 points)).

Soit A la formule suivante : $\neg\forall x P(x) \vee \neg\forall x Q(x) \Rightarrow \neg(\forall x P(x) \wedge \forall x Q(x))$

Montrez que A est valide en montrant que $\neg A$ est contradictoire par la méthode de Herbrand, en suivant les étapes suivantes :

1. (10 points) *Skolémisez $\neg A$ puis mettez-la en forme clausale.*
2. (5 points) *Proposez des instances contradictoires des clauses obtenues et montrez la contradiction par résolution propositionnelle.*

□

Exercice 5 (Résolution (15 points)).

Considérons l'ensemble suivant de clauses du premier ordre :

$$\Gamma \;=\; \{\overline{C(x,y)} + \overline{E(z,x)} + E(z,y),\ C(c,a),\ C(a,b),\ \overline{C(c,b)},\ C(x,y) + E(x,x),\ C(x,y) + \overline{E(x,y)}\}$$

Nous voulons montrer que cet ensemble de clauses est insatisfaisable par instanciation et par résolution.

1. (7 points) *Trouvez des instances contradictoires de ces clauses et montrez par résolution propositionnelle que ces instances sont contradictoires.*
2. (8 points) *Donnez une preuve directe du caractère contradictoire de Γ par factorisation, copie et résolution binaire. Il est possible que toutes les règles ne soient pas utilisées.*

□

Exercice 6 (Déduction naturelle (20 points)).

Démontrez les formules suivantes en déduction naturelle au premier ordre. Comme vu en cours et en TD, on numérotera les lignes de la preuve, et on indiquera pour chaque ligne la formule démontrée, le contexte, la règle appliquée et les prémisses utilisées.

1. (10 points) $\neg\exists x(P(x) \vee Q(x)) \Rightarrow \forall x(\neg P(x) \wedge \neg Q(x))$
2. (10 points) $\forall x \forall y(\exists z(x = z \wedge z = y) \Rightarrow (x = y))$

□

Exercice 7 (Démonstration par récurrence (15 points)).

Soit n un entier positif non nul. Soient $A_1, A_2, \ldots A_n$ et B des formules du premier ordre.

On suppose que x_1 est la seule variable libre dans A_1, que x_2 est la seule variable libre dans A_2, etc. et que y est la seule variable libre dans B.

On s'intéresse à la formule $\exists x_n A_n \Rightarrow \ldots \Rightarrow \exists x_2 A_2 \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B$.

1. (1 point) *Rappeler quelle est la formule stricte (parenthésée) correspondant à cette formule.*
2. (14 points) *Démontrer que, pour tout $n \geqslant 1$, on a*

$$\exists x_n A_n \Rightarrow \ldots \Rightarrow \exists x_2 A_2 \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B \quad \equiv \quad \forall x_n \ldots \forall x_2 \forall x_1 \forall y(A_n \Rightarrow \ldots \Rightarrow A_2 \Rightarrow A_1 \Rightarrow B)$$

□

Corrigés

Exercice 1, page 1

1. La 1-expansion est valide : $P(0) \Rightarrow (P(0) \Rightarrow 0 = 0)$
   La 2-expansion est :

$$\begin{aligned} & \big(P(0) \Rightarrow (P(0) \Rightarrow 0 = 0).(P(1) \Rightarrow 0 = 1)\big) + \big(P(1) \Rightarrow (P(0) \Rightarrow 1 = 0).(P(1) \Rightarrow 1 = 1)\big) \\ \equiv \quad & \big(P(0) \Rightarrow (P(1) \Rightarrow \bot)\big) + \big(P(1) \Rightarrow (P(0) \Rightarrow \bot)\big) \\ \equiv \quad & (P(0) \Rightarrow \overline{P(1)}) + (P(1) \Rightarrow \overline{P(0)}) \;\equiv\; P(0) \Rightarrow \overline{P(1)} \end{aligned}$$

   Le (seul) contre-modèle est donc $P_I = \{0, 1\}$.
2. La 1-expansion est valide : $R(0,0) \Rightarrow R(0,0)$
   La 2-expansion est :
   $(R(0,0) + R(0,1)).(R(1,0) + R(1,1)) \Rightarrow R(0,0).R(1,0) + R(0,1).R(1,1)$.
   Les contre-modèles sont donc $R_I = \{(0,0),\ (1,1)\}$ et $R_I = \{(0,1),\ (1,0)\}$.

Exercice 2, page 1

1. Tout frère du père de Robert est un oncle de Robert :

$$\forall x(f(x, p(r)) \Rightarrow o(x, r)).$$

2. Si les pères de deux enfants sont des frères alors ces deux enfants sont des cousins :

$$\forall x \forall y(f(p(x), p(y)) \Rightarrow c(x, y)).$$

3. Robert a un cousin plus jeune qu'un des frères de Robert :

$$\exists x(c(x, r) \wedge \exists y(f(y, r) \wedge j(x, y))).$$

1. $\neg\exists x\ j(p(x), x)$: personne n'est plus âgé que son père
2. $\forall x \forall y(p(p(x)) = p(p(y)) \Rightarrow c(x, y))$: deux personnes qui ont le même grand-père paternel sont cousines
3. $\forall x \exists y(f(x, y) \wedge j(x, y))$: tout le monde a un frère aîné (impossible mais ça n'empêche pas de l'exprimer...)
4. $\exists x \exists y(f(x, y) \wedge \neg(p(x) = p(y)))$: il y a deux frères qui n'ont pas le même père

Exercice 3, page 1

1. $f(x, g(a), x) = f(y, g(y), b)$
   Décomposition : $x = y$, $g(a) = g(y)$, $x = b$
   Élimination de x : $x := y$, $g(a) = g(y)$, $y = b$
   Décomposition : $x := y$, $a = y$, $y = b$
   Élimination de y : $x := y$, $a = b$, $y := b$
   Échec de décomposition : pas de solution.
2. $f(f(x, y, a), g(y), h(y, b)) = f(z, x, h(b, b))$
   Décomposition : $f(x, y, a) = z$, $g(y) = x$, $h(y, b) = h(b, b)$
   Orientation ($\times 2$) : $z = f(x, y, a)$, $x = g(y)$, $h(y, b) = h(b, b)$
   Élimination de z : $z := f(x, y, a)$, $x = g(y)$, $h(y, b) = h(b, b)$
   Élimination de x : $z := f(g(y), y, a)$, $x := g(y)$, $h(y, b) = h(b, b)$
   Décomposition : $z := f(g(y), y, a)$, $x := g(y)$, $y = b$, $b = b$

Identité : $z := f(g(y), y, a)$, $x := g(y)$, $y = b$

Élimination de y : $z := f(g(b), b, a)$, $x := g(b)$, $y := b$

$< x := g(b), y := b, z := f(g(b), b, a) >$ est une solution la plus générale de l'équation.

3. $h(g(x), h(y, x)) = h(g(y), h(g(z), z))$

   Décomposition : $g(x) = g(y)$, $h(y, x) = h(g(z), z))$

   Décomposition : $x = y$, $h(y, x) = h(g(z), z))$

   Élimination de x : $x := y$, $h(y, y) = h(g(z), z))$

   Décomposition : $x := y$, $y = g(z)$, $y = z$

   Élimination de y : $x := y$, $y := g(z)$, $g(z) = z$

   Orientation : $x := y$, $y := g(z)$, $z = g(z)$

   Échec d'élimination de z : pas de solution.

Exercice 4, page 2

$\neg A = \neg(\neg\forall x P(x) \vee \neg\forall x Q(x) \Rightarrow \neg(\forall x P(x) \wedge \forall x Q(x)))$

1. Forme normale : $(\exists x \neg P(x) \vee \exists x \neg Q(x)) \wedge \forall x P(x) \wedge \forall x Q(x))$
2. Forme propre : $(\exists x \neg P(x) \vee \exists y \neg Q(y)) \wedge \forall u P(u) \wedge \forall v Q(v))$
3. Elimination des $\exists$: $(\neg P(a) \vee \neg Q(b)) \wedge \forall u P(u) \wedge \forall v Q(v))$
4. Enlèvement des $\forall$: $(\neg P(a) \vee \neg Q(b)) \wedge P(u) \wedge Q(v))$
5. Forme clausale : $\{\neg P(a) \vee \neg Q(b), P(u), Q(v)\}$
6. Instances contradictoires : $\neg P(a) \vee \neg Q(b), P(a), Q(b)$

1 :	$\neg P(a) \vee \neg Q(b)$	Hyp
2 :	$P(a)$	Hyp
3 :	$\neg Q(b)$	Res 1,2
4 :	$Q(b)$	Hyp
5 :	$\perp$	Res 3,4

Exercice 5, page 2

Instanciation : Nous obtenons les instances contradictoires suivantes :

$$\begin{aligned}\Gamma \quad = \quad & \{\overline{C(c,a)} + \overline{E(c,c)} + E(c,a), \overline{C(a,b)} + \overline{E(c,a)} + E(c,b), \quad \text{(deux instances)} \\ & C(c,a), C(a,b), \overline{C(c,b)}, E(c,c) + C(c,b), \overline{E(c,b)} + C(c,b)\}\end{aligned}$$

Par résolution propositionnelle nous avons la preuve :

(1)	$\overline{C(c,b)}$	Hyp
(2)	$E(c,c) + C(c,b)$	Hyp
(3)	$\overline{E(c,b)} + C(c,b)$	Hyp
(4)	$E(c,c)$	Res 1,2
(5)	$\overline{E(c,b)}$	Res 1,3
(6)	$\overline{C(c,a)} + \overline{E(c,c)} + E(c,a)$	Hyp
(7)	$\overline{C(a,b)} + \overline{E(c,a)} + E(c,b)$	Hyp
(8)	$\overline{C(c,a)} + E(c,a)$	Res 4,6
(9)	$\overline{C(a,b)} + \overline{E(c,a)}$	Res 5,7
(10)	$\overline{C(c,a)} + \overline{C(a,b)}$	Res 8,9
(11)	$C(c,a)$	Hyp
(12)	$\overline{C(a,b)}$	Res 10,11
(13)	$C(a,b)$	Hyp
(14)	$\perp$	Résolvant 12,13

Résolution binaire :

numéro	conclusion	justification	unificateur
(1)	$\overline{C(c,b)}$	Hyp	
(2)	$\overline{E(x,x)} + C(x,y)$	Hyp	
(3)	$\overline{E(x,y)} + C(x,y)$	Hyp	
(4)	$\overline{E(c,c)}$	Res 1,2	$< x := c, y := b >$
(5)	$\overline{E(c,b)}$	Res 1,3	$< x := c, y := b >$
(6)	$\overline{C(x,y)} + \overline{E(z,x)} + E(z,y)$	Hyp	
(7)	$\overline{C(c,y)} + \overline{E(c,y)}$	Res 4,6	$< x := c, z := c >$
(8)	$\overline{C(x,b)} + \overline{E(c,x)}$	Res 5,6	$< y := b, z := c >$
(9)	$\overline{C(c,y)} + \overline{C(y,b)}$	Res 7,8	$< x := y >$
(10)	$\overline{C(c,a)}$	Hyp	
(11)	$\overline{C(a,b)}$	Res 9,10	$< y := a >$
(12)	$C(a,b)$	Hyp	
(13)	$\perp$	Résolvant 11,12	

Exercice 6, page 2

1. $\neg\exists x(P(x) \vee Q(x)) \Rightarrow \forall x(\neg P(x) \wedge \neg Q(x))$

contexte	numéro	preuve	justification
1	1	`Supposons` $\neg\exists x(P(x) \vee Q(x))$	
1,2	2	`Supposons` $P(x)$	
1,2	3	$P(x) \vee Q(x)$	$\vee I1$ 2
1,2	4	$\exists x(P(x) \vee Q(x))$	$\exists I$ 3 x
1,2	5	$\perp$	$\Rightarrow E$ 1, 4
1	6	`Donc` $\neg P(x)$	$\Rightarrow I$ 2, 5
1,7	7	`Supposons` $Q(x)$	
1,7	8	$P(x) \vee Q(x)$	$\vee I2$ 7
1,7	9	$\exists x(P(x) \vee Q(x))$	$\exists I$ 8 x
1,7	10	$\perp$	$\Rightarrow E$ 1, 9
1	11	`Donc` $\neg Q(x)$	$\Rightarrow I$ 7, 10
1	12	$\neg P(x) \wedge \neg Q(x)$	$\wedge I$ 6, 11
1	13	$\forall x(\neg P(x) \wedge \neg Q(x))$	$\forall I$ 12
	14	`Donc` $\neg\exists x(P(x) \vee Q(x)) \Rightarrow \forall x(\neg P(x) \wedge \neg Q(x))$	$\Rightarrow I$ 1, 13

2. $\forall x \forall y(\exists z(x = z \wedge z = y) \Rightarrow (x = y))$

contexte	numéro	preuve	règle
1	1	`Supposons` $\exists z(x = z \wedge z = y)$	
1,2	2	`Supposons` $(x = z \wedge z = y)$	
1,2	3	$x = z$	$\wedge$E1 2
1,2	4	$z = y$	$\wedge$E2 2
1,2	5	$x = y$	congruence 4,3
1	6	`Donc` $(x = z \wedge z = y) \Rightarrow (x = y)$	$\Rightarrow$I 2,5
1	7	$x = y$	$\exists$E 1,6
	8	`Donc` $\exists z(x = z \wedge z = y) \Rightarrow (x = y)$	$\Rightarrow$I 1,7
	9	$\forall y(\exists z(x = z \wedge z = y) \Rightarrow (x = y))$	$\forall$I 8
	10	$\forall x \forall y(\exists z(x = z \wedge z = y) \Rightarrow (x = y))$	$\forall$I 9

3.

Exercice 7, page 2

1. $\Big(\exists x_n A_n \Rightarrow \Big(\ldots \Rightarrow \big(\exists x_2 A_2 \Rightarrow (\exists x_1 A_1 \Rightarrow \forall y B)\big)\Big)\Big)$.
2. Par récurrence (directe) sur n :

— Pour $n = 0$ on a trivialement $\forall y B \equiv \forall y B$
— Supposons $n \geq 1$ et la propriété vraie pour $n - 1$. Alors :

$$\begin{aligned}
\exists x_n A_n \Rightarrow \ldots \Rightarrow \exists x_2 A_2 \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B \quad &\equiv \quad \neg(\exists x_n A_n) \vee (\exists x_{n-1} A_{n-1} \Rightarrow \ldots \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B) \\
&\equiv \quad (\forall x_n \neg A_n) \vee (\exists x_{n-1} A_{n-1} \Rightarrow \ldots \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B)
\end{aligned}$$

Or la formule $\exists x_{n-1} A_{n-1} \Rightarrow \ldots \Rightarrow \exists x_1 A_1 \Rightarrow \forall y B$ satisfait la propriété au rang $n - 1$:

$$\equiv \quad (\forall x_n \neg A_n) \vee \forall x_{n-1} \ldots \forall x_1 \forall y (A_{n-1} \Rightarrow \ldots \Rightarrow A_1 \Rightarrow B)$$

Comme les variables ne sont pas libres dans les autres parties de la formule :

$$\begin{aligned}
&\equiv \quad \forall x_n \forall x_{n-1} \ldots \forall x_1 \forall y \big(\neg A_n \vee (A_{n-1} \Rightarrow \ldots \Rightarrow A_1 \Rightarrow B)\big) \\
&\equiv \quad \forall x_n \forall x_{n-1} \ldots \forall x_1 \forall y (A_n \Rightarrow A_{n-1} \Rightarrow \ldots \Rightarrow A_1 \Rightarrow B)
\end{aligned}$$

Partiel INF452

Stéphane Devismes Benjamin Wack

10 mars 2021
2 pages
Total : 120 points + 5 points de bonus
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices.
L'épreuve sera notée sur 120 points + 5 points de bonus.
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Modèles et formes normales (20 points))
Soit $A = a \wedge d \Rightarrow b \wedge c \Leftrightarrow \neg c \Rightarrow \neg d \wedge a$.

1. *Réécrivez cette formule sous forme d'arbre.* *(1 point)*
2. *Déterminer une forme normale disjonctive de A et la simplifier le plus possible.* *(8 points)*
3. *Déterminer une forme normale conjonctive de A et la simplifier le plus possible.* *(8 points)*
4. *Si c'est possible, donner un modèle de A et justifier la réponse* sans utiliser de table de vérité. *(1,5 points)*
5. *Si c'est possible, donner un contre-modèle de A et justifier la réponse* sans utiliser de table de vérité. *(1,5 points)*

Réponse:
— Je donne A sous sa forme stricte : $(((a \wedge d) \Rightarrow (b \wedge c)) \Leftrightarrow (\neg c \Rightarrow (\neg d \wedge a)))$.
— En forme normale disjonctive : $a \wedge \neg d \vee a \wedge b \vee a \wedge \neg c \vee \neg a \wedge c$
Ce qui donne quatre modèles : (1) $a = 1, d = 0$, (2) $a = 1, b = 1$, (3) $a = 1, c = 0$ et (4) $a = 0$ et $c = 1$.
— En forme normale conjonctive : $(\neg a \vee b \vee \neg c \vee \neg d) \wedge (a \vee c)$
Ce qui donne deux contre-modèles : (1) $a = 1, b = 0, c = 1, d = 1$ et (2) $a = 0, c = 0$.

□

Exercice 2 (Formalisation (25 points))

Nous retournons sur l'île des Purs et des Pires[1]*. Rappelons que les Purs disent toujours la vérité tandis que les Pires mentent toujours.*

Nous rencontrons cette fois trois indigènes, Aha, Bébé et Sissi. Nous savons que l'un d'entre eux est Pur alors que les deux autres sont Pires. Ils affirment :

Aha : *« Il n'y a aucun danger sur cette île. »*

Bébé : *« Aha est Pur. »*

Sissi : *« Bébé est Pire. »*

Peut-on en conclure que l'île est dangereuse ?

1. *Formaliser les trois affirmations, la conclusion, et le fait qu'il y a un Pur et deux Pires. (10 points)*
Pour cela, vous utiliserez les variables propositionnelles suivantes :
— *A : Aha est Pur.*
— *B : Bébé est Pur.*
— *C : Sissi est Pur.*

1. Cette énigme est librement inspirée d'un problème de Jean-Claude Baillif et réutilise les personnages de Raymond Smullyan.

— *D : L'île est dangereuse.*

2. *Transformer chacune des trois affirmations, le fait qu'il y a un Pur et deux Pires, ainsi que la négation de la conclusion en un ensemble de clauses équivalentes. (10 points)*
3. *Démontrer,* **par résolution**, *que la conclusion se déduit des trois affirmations. (5 points)*

Réponse:

1. **Affirmation de Aha :** $A \Leftrightarrow \neg D$

   Affirmation de Bébé : $B \Leftrightarrow A$

   Affirmation de Sissi : $C \Leftrightarrow \neg B$

   Conclusion : D

   Tribus des personnages : par exemple $A \wedge \neg B \wedge \neg C + \neg A \wedge \neg B \wedge C + \neg A \wedge B \wedge \neg C$
2. Clauses :
   — $\bar{A} + \bar{D}$, $A + D$
   — $\bar{B} + A$, $\bar{A} + B$
   — $\bar{B} + \bar{C}$, $B + C$
   — $\bar{D}$
   — $A + B + C$, $\bar{A} + \bar{B}$, $\bar{B} + \bar{C}$, $\bar{A} + \bar{C}$
3. On n'utilise en fait que très peu de clauses, et Sissi ne nous apprend rien :

```
1 -D      hyp
2 A+D     hyp
3 A       res 1,2
4 -A+B    hyp
5 -A+-B   hyp
6 -A      res 4,5
7 Faux    res 3,6
```

□

Exercice 3 (Résolution (Exo du Poly, 20 points))

Les ensembles de formules suivants sont insatisfaisables.

— $\{a + b, \overline{a} + c, \overline{a} + \overline{d}, d + \overline{c}, \overline{b} + a\}$.

— $\{a + b + c, \overline{a} + b, \overline{b} + c, \overline{c} + a, \overline{a} + \overline{b} + \overline{c}\}$.

En donner une preuve par résolution.

Réponse:

— Soit l'ensemble de clauses : $\{a + b, \overline{a} + c, \overline{a} + \overline{d}, d + \overline{c}, \overline{b} + a\}$. Nous suggérons une tactique pour obtenir une preuve : construire en priorité les résolvants ayant le moins de littéraux possibles.

1	$a + b$	Hyp
2	$\overline{b} + a$	Hyp
3	a	Résolvant de 1, 2
4	$\overline{a} + c$	Hyp
5	c	Résolvant de 3, 4
6	$\overline{a} + \overline{d}$	Hyp
7	$\overline{d}$	Résolvant de 3, 6
8	$d + \overline{c}$	Hyp
9	d	Résolvant de 5, 8
10	$\perp$	Résolvant de 7,9

— Soit l'ensemble de clauses : $\{a + b + c, \overline{a} + b, \overline{b} + c, \overline{c} + a, \overline{a} + \overline{b} + \overline{c}\}$. Nous utilisons la même tactique pour obtenir une preuve.

1	$a+b+c$	Hyp
2	$\overline{a}+b$	Hyp
3	$b+c$	Résolvant de 1,2
4	$\overline{c}+a$	Hyp
5	$b+\overline{c}$	Résolvant de 2, 4
6	b	Résolvant de 3, 5
7	$\overline{b}+c$	Hyp
8	c	Résolvant de 6, 7
9	a	Résolvant de 4, 8
10	$\overline{a}+\overline{b}+\overline{c}$	Hyp
11	$\overline{b}+\overline{c}$	9, 10
12	$\overline{c}$	Résolvant de 6, 11
13	$\perp$	Résolvant de 8, 12

□

Exercice 4 (DPLL (20 points))

Appliquer l'algorithme `DPLL` *sur chacun des ensembles de clauses suivants. Vous donnerez la trace de l'algorithme sous forme arborescente en étiquetant clairement chaque étape par la règle utilisée et les assignations qui en découlent. Vous interpréterez clairement les résultats obtenus lorsque l'algorithme se termine et vous donnerez un modèle le cas échéant.*

1. $\{a+\overline{b}+c,\ \overline{a}+b,\ \overline{a}+\overline{c},\ a+\overline{c}+\overline{d},\ \overline{a}+c+\overline{d},\ a,\ \overline{c},\ \overline{b}+d\}$ *(5 points)*
2. $\{a+b+c+\overline{d}+\overline{e}+\overline{f},\ \overline{b}+c,\ a+c+d+f,\ \overline{a}+\overline{d}+\overline{e},\ \overline{b}+\overline{c}+\overline{f},\ \overline{b}+\overline{d}+e,\ b+\overline{c},\ b+\overline{c}+\overline{d},\ b+c+d,\ b+\overline{d},\ b+e,\ \overline{f}\}$ *(15 points)*

Réponse:

1. `a+-b+c, -a+b, -a+-c, a+-c+-d, -a+c+-d, a, -c, -b+d`

   `RE`

   `-a+b, -a+c+-d, a, -c, -b+d`

   `RU a=1, c=0`

   `b, -d, -b+d`

   `RU b=1, d=0`

   `Faux`

2. `a+b+c+-d+-e+-f, -b+c, a+c+d+f, -a+-d+-e, -b+-c+-f, -b+-d+e,`
   `b+-c, b+-c+-d, b+c+d, b+-d, b+e, -f`

   `RE`

   `-b+c, a+c+d+f, -a+-d+-e, -b+-d+e, b+-c, b+c+d, b+-d, b+e, -f`

   `RU f=0`

   `-b+c, a+c+d, -a+-d+-e, -b+-d+e, b+-c, b+c+d, b+-d, b+e`

   `Branchement sur b`

   `1) b=0`

   `a+c+d, -a+-d+-e, -c, c+d, -d, e`

```
RE

-c, c+d, -d, e

ELI e=1

-c, c+d, -d

RU c=0, d=0

FAUX

2) b=1

c, a+c+d, -a+-d+-e, -d+e

RE

c, -a+-d+-e, -d+e

ELI c=1

-a+-d+-e, -d+e

ELI a=0

-d+e

ELI d=0

Vrai
```

D'où, le modèle $a = 0, b = 1, c = 1, d = 0, f = 0$.

□

Exercice 5 (Déduction naturelle (25 points))
Prouver, **par déduction naturelle**, *les formules suivantes. Vous présenterez vos preuves sous forme de tableaux.*

1. $(C \Rightarrow \neg B) \wedge (A \Rightarrow B) \Rightarrow (C \Rightarrow \neg A)$ *(10 points)*
2. $A \wedge (\neg A \vee B) \Rightarrow A \wedge B$ *(10 points)*
3. $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$ *(5 points)*

Réponse:

1. $(C \Rightarrow \neg B) \wedge (A \Rightarrow B) \Rightarrow (C \Rightarrow \neg A)$

```
assume (c => -b) & (a => b).
  assume c.
    assume a.
      c => -b.
      a => b.
      -b.
      b.
      F.
    therefore -a.
  therefore c => -a.
therefore (c => -b) & (a => b) => c => -a.
```

2. $A \wedge (\neg A \vee B) \Rightarrow A \wedge B$

```
assume a & (-a + b).
  a.
  -a + b.
  assume -a.
    F.
    a & b.
  therefore -a => a & b.
  assume b.
    a & b.
  therefore b => a & b.
  a & b.
therefore a & (-a + b) => a & b.
```

3. $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$

```
assume -a => -b.
  assume b.
    assume -a.
      -b.
      F.
    therefore --a.
    a.
  therefore b => a.
therefore (-a => -b) => b => a.
```

□

Exercice 6 (Démonstration par récurrence (15 points))

Dans tout cet exercice on note $\oplus$ l'opérateur « ou exclusif », dont nous rappelons la table de vérité :

x	y	$x \oplus y$
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

— *Démontrer que le connecteur $\oplus$ est associatif, autrement dit que $(x \oplus y) \oplus z \equiv x \oplus (y \oplus z)$.*

— *Soient maintenant n variables* différentes $x_1 \dots x_n$.
Démontrer par récurrence que pour toute assignation v, on a $[x_1 \oplus x_2 \dots \oplus x_n]_v = 1$ si et seulement si le nombre de ces variables valant 1 dans v est impair.

Réponse:

— Par les tables de vérité, il apparaît clairement que les modèles de ces deux formules sont les mêmes : les assignations dans lesquelles une seule ou les trois variables valent 1.

— **Cas de base :** Si $n = 1$, on a une seule variable et la propriété est trivialement vérifiée. On peut aussi commencer à $n = 2$ sans difficulté.

Hypothèse de récurrence : Supposons que pour $n \geq 1$, on a $[x_1 \oplus \dots \oplus x_n]_v = 1$ si et seulement si le nombre de $[x_i]_v$ (i allant de 1 à n) valant 1 est impair.

Pas de récurrence Considérons $n+1$ variables différentes : $x_1 \dots x_{n+1}$.

On a 4 cas a étudier :

1. $[x_1 \oplus \dots \oplus x_n]_v = 0$ et $[x_{n+1}]_v = 0$. Dans ce cas, $[x_1 \oplus \dots \oplus x_{n+1}]_v = 0 \oplus 0 = 0$. Or, par hypothèse de récurrence, le nombre de $[x_i]_v$ (i allant de 1 à n) valant 1 est pair, ce qui reste vrai pour i de 1 à $n+1$ puisque $[x_{n+1}]_v = 0$.
2. $[x_1 \oplus \dots \oplus x_n]_v = 0$ et $[x_{n+1}]_v = 1$. Dans ce cas, $[x_1 \oplus \dots \oplus x_{n+1}]_v = 0 \oplus 1 = 1$. Or, par hypothèse de récurrence, le nombre de $[x_i]_v$ (i allant de 1 à n) valant 1 est pair, et donc il devient impair pour i de 1 à $n+1$.

3. $[x_1 \oplus \ldots \oplus x_n]_v = 1$ et $[x_{n+1}]_v = 0$. Dans ce cas, $[x_1 \oplus \ldots \oplus x_{n+1}]_v = 1 \oplus 0 = 1$. Or, par hypothèse de récurrence, le nombre de $[x_i]_v$ (i allant de 1 à n) valant 1 est impair, ce qui reste vrai pour i de 1 à $n+1$.
4. $[x_1 \oplus \ldots \oplus x_n]_v = 1$ et $[x_{n+1}]_v = 1$. Dans ce cas, $[x_1 \oplus \ldots \oplus x_{n+1}]_v = 1 \oplus 1 = 0$. Or, par hypothèse de récurrence, le nombre de $[x_i]_v$ (i allant de 1 à n) valant 1 est impair, et donc il devient pair pour i de 1 à $n+1$.

$\square$

Examen INF452

Stéphane Devismes Benjamin Wack

mai 2021
2 pages
Total : 120 points
Durée : 2h00

Documents autorisés : une feuille recto verso de notes manuscrites format A4.
Le barème est *indicatif*, les points correspondent au nombre de minutes nécessaires pour réaliser les exercices. L'épreuve sera notée sur 120 points.
Le résultat d'une question peut être admis pour s'en servir dans la suite de l'énoncé. Les exercices peuvent être traités dans l'ordre de votre choix à condition de les numéroter clairement.

Exercice 1 (Expansions (20 points))**.**

1. (10 points) *À l'aide de la méthode des expansions, déterminer* ***un contre-modèle*** *de la formule* $A = \forall x \forall y (R(x,y) \wedge R(y,x) \Rightarrow R(x,x))$.
2. (10 points) *À l'aide de la méthode des expansions, déterminer* ***un modèle*** *de la formule* $B = \forall x \exists y (P(x,y) \wedge x \neq y)$.

□

Exercice 2 (Formalisation (20 points, d'après un exercice de Christine Paulin))**.**

On cherche à modéliser un plan de table, c'est-à-dire comment les personnes sont installées pour le repas. On utilisera les symboles suivants :

— $Voisin(x,y)$ *signifie que* x *est assis à côté de* y *;*
— $Ami(x,y)$ *signifie que* x *et* y *sont amis ;*
— $Femme(x)$ *signifie que* x *est une femme.*

1. (15 points) *À l'aide de ces symboles, exprimez par des formules du premier ordre les énoncés suivants :*
   (a) *Les personnes qui sont assises côte-à-côte sont toutes amies.*
   (b) *Deux hommes ne sont jamais assis côte-à-côte.*
   (c) *Chaque femme est assise à côté d'au moins un homme.*
   (d) *Personne n'est assis tout seul.*
   (e) *On ne peut pas être assis à côté de soi-même.*
   (f) *Personne n'est assis auprès de plus de deux personnes.*
2. (5 points) *Soit le domaine* $D = \{0,1,2,3,4\}$. *On suppose que* 3 *et* 4 *sont les seules femmes. Tout le monde est ami avec tout le monde, sauf* 0 *et* 3 *qui se détestent.*

   Proposez une interprétation de la relation $Voisin$ *qui donne un modèle de l'ensemble des formules précédentes.*

□

Exercice 3 (Unification (15 points)).
Pour chacune des équations suivantes, déterminez si elle admet une solution, et si oui déterminez un unificateur le plus général. Vous utiliserez pour cela l'algorithme du cours dont vous détaillerez les étapes en donnant à chaque fois le nom des règles utilisées.

1. (5 points) $h(a, g(x, b), f(y)) = h(x, g(a, y), f(y))$
2. (5 points) $h(x, y, g(x, b)) = h(y, z, g(f(z), b))$
3. (5 points) $h(x, y, g(x, y)) = h(z, z, g(g(a, b), f(a)))$

□

Exercice 4 (Skolémisation (15 points)).
Soit A la formule suivante : $\forall x(Q(x) \Rightarrow R(x)) \wedge \exists x(P(x) \wedge Q(x)) \Rightarrow \exists x(P(x) \wedge R(x))$
Montrez que A est valide en montrant que $\neg A$ est contradictoire par la méthode de Herbrand, en suivant les étapes suivantes :

1. (10 points) *Skolémisez $\neg A$ puis mettez-la en forme clausale.*
2. (5 points) *Proposez des instances contradictoires des clauses obtenues et montrez la contradiction par résolution propositionnelle.*

□

Exercice 5 (Résolution (15 points)).
Considérons l'ensemble suivant de clauses du premier ordre :

$$\Gamma \;=\; \{R(x,y) + R(f(x),x), \quad R(x,y) + R(x,f(x)), \quad \overline{R(x,y)} + \overline{R(y,z)} + R(x,z), \quad \overline{R(y,y)}\}$$

Nous voulons montrer que cet ensemble de clauses est insatisfaisable par instanciation et par résolution.

1. (7 points) *Trouvez des instances contradictoires de ces clauses et montrez par résolution propositionnelle que ces instances sont contradictoires.*
2. (8 points) *Donnez une preuve directe du caractère contradictoire de Γ par factorisation, copie et résolution binaire. Il est possible que toutes les règles ne soient pas utilisées.*

□

Exercice 6 (Déduction naturelle (20 points)).
Démontrez les formules suivantes en déduction naturelle au premier ordre. Comme vu en cours et en TD, on numérotera les lignes de la preuve, et on indiquera pour chaque ligne la formule démontrée, le contexte, la règle appliquée et les prémisses utilisées.

1. (10 points) $\forall x \forall y \forall z(x = y \wedge \neg(y = z) \Rightarrow \neg(x = z))$
2. (10 points) $\exists x P(x) \wedge \forall x \forall y(P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x(P(x) \wedge \forall y(P(y) \Rightarrow x = y))$

□

Exercice 7 (Connecteur complet (15 points, d'après un exercice du poly)).
Soit $\mid$ l'opération suivante : $x \mid y = 1$ si et seulement si $x = 0$ et $y = 0$. Cette opération est aussi appelée ni *car $x \mid y$ est vrai si et seulement si ni x, ni y ne sont vrais.*
L'objectif de cet exercice est de démontrer que ce connecteur est complet.

1. (2 points) *Donner une forme normale conjonctive et une forme normale disjonctive de la formule $x \mid y$.*
2. (2 points) *Ce connecteur est-il commutatif ($x \mid y \equiv y \mid x$) ?*
   Est-il est associatif ($(x \mid y) \mid z \equiv x \mid (y \mid z)$) ?
3. (1 point) *Donner la table de vérité de la formule $x \mid x$.*
4. (10 points) *Donner des formules équivalentes à chacune des formules : $\top$, $\bot$, $\neg x$, $x \wedge y$, $x \vee y$ et $x \Rightarrow y$ en n'utilisant que le connecteur $\mid$ et possiblement les variables x et y. On pourra justifier le résultat soit par des tables de vérité, soit en utilisant des équivalences avec les formules propositionnelles usuelles.*

□

Corrigés

Exercice 1, page 1

1. Contre-modèle par expansion :
   $D = \{0\}$: $R(0,0).R(0,0) \Rightarrow R(0,0)$ valide
   $D = \{0,1\}$: $(R(0,0).R(0,0) \Rightarrow R(0,0)).(R(0,1).R(1,0) \Rightarrow R(0,0)).(R(1,0).R(0,1) \Rightarrow R(1,1)).(R(1,1).R(1,1) \Rightarrow R(1,1))$
   Tout R_I contenant les deux couples $(0,1)$ et $(1,0)$, et faux pour au moins un couple (i,i) convient.
2. Modèle par expansion :
   $D = \{0\}$: $P(0,0).(0 \neq 0)$ est insatisfaisable.
   $D = \{0,1\}$: $(P(0,0).(0 \neq 0) + P(0,1).(0 \neq 1)).(P(1,0).(1 \neq 0) + P(1,1).(1 \neq 1)) \equiv P(0,1).P(1,0)$
   Tout P_I contenant $(0,1)$ et $(1,0)$ convient.

Exercice 2, page 1

1. Note : d'autres solutions sont admissibles, notamment selon la façon dont la symétrie des relations $Voisin$ et Ami est prise en compte.
   (a) Les personnes qui sont assises côte-à-côte sont toutes amies. $\forall x \forall y (Voisin(x,y) \Rightarrow Ami(x,y))$
   (b) Deux hommes ne sont jamais assis côte-à-côte. $\forall x \forall y (Voisin(x,y) \Rightarrow Femme(x) \vee Femme(y))$
   (c) Chaque femme est assise à côté d'au moins un homme. $\forall x (Femme(x) \Rightarrow \exists y (Voisin(x,y) \wedge \neg Femme(y)))$
   (d) Personne n'est assis tout seul. $\forall x \exists y Voisin(x,y)$
   (e) On ne peut pas être assis à côté de soi-même. $\forall x \forall y (Voisin(x,y) \Rightarrow \neg x = y)$
   (f) Personne n'est assis auprès de plus de deux personnes. $\forall x \forall y_1 \forall y_2 \forall y_3 (Voisin(x,y_1) \wedge Voisin(x,y_2) \wedge Voisin(x,y_3) \wedge \neg y_1 = y_2 \Rightarrow y_3 = y_1 \vee y_3 = y_2)$
2. On a 2 femmes et 3 hommes, comme deux hommes ne peuvent être côte-à-côte, on alterne homme-femme-homme-femme-homme. Comme 0 et 3 se détestent, la femme 3 doit être entre 1 et 2. On a donc $Voisin_I = \{(0,4),(4,1),(1,3),(3,2),(2,3),(3,1),(1,4),(4,0)\}$.
   Si on accepte 2 tables séparées, d'autres solutions du type homme-femme-homme et femme-homme sont possibles.

Exercice 3, page 2

1. $h(a, g(x,b), f(y)) = h(x, g(a,y), f(y))$
   Décomposition : $a = x$, $g(x,b) = g(a,y)$, $f(y) = f(y)$
   Identité : $a = x$, $g(x,b) = g(a,y)$
   Décomposition : $a = x$, $x = a$, $b = y$
   Élimination de x : $a = a$, $x := a$, $b = y$
   Identité : $x := a$, $b = y$
   Orientation : $x := a$, $y = b$
   Élimination de y : $x := a$, $y := b$
   $< x := a, y := b >$ est une solution la plus générale de l'équation.
2. $h(x, y, g(x,b)) = h(y, z, g(f(z), b))$
   Décomposition : $x = y$, $y = z$, $g(x,b) = g(f(z), b)$
   Élimination de x : $x := y$, $y = z$, $g(y,b) = g(f(z), b)$
   Élimination de y : $x := z$, $y := z$, $g(z,b) = g(f(z), b)$
   Décomposition : $x := z$, $y := z$, $z = f(z)$, $b = b$
   Échec de l'élimination de z pas de solution

3. $h(x, y, g(x, y)) = h(z, z, g(g(a, b), f(a)))$
   Décomposition : $x = z$, $y = z$, $g(x, y) = g(g(a, b), f(a))$
   Élimination de x : $x := z$, $y = z$, $g(z, y) = g(g(a, b), f(a))$
   Élimination de y : $x := z$, $y := z$, $g(z, z) = g(g(a, b), f(a))$
   Décomposition : $x := z$, $y := z$, $z = g(a, b)$, $z = f(a)$
   Élimination de z : $x := g(a, b)$, $y := g(a, b)$, $z := g(a, b)$, $g(a, b) = f(a)$
   Échec de la décomposition : pas de solution

Exercice 4, page 2

1. Forme normale : $\forall x(\neg Q(x) \vee R(x)) \wedge \exists x(P(x) \wedge Q(x)) \wedge \forall x(\neg P(x) \vee \neg R(x))$
2. Forme propre : $\forall x(\neg Q(x) \vee R(x)) \wedge \exists y(P(y) \wedge Q(y)) \wedge \forall z(\neg P(z) \vee \neg R(z))$
3. Elimination des $\exists$: $\forall x(\neg Q(x) \vee R(x)) \wedge P(a) \wedge Q(a) \wedge \forall z(\neg P(z) \vee \neg R(z))$
4. Enlèvement des $\forall$: $(\neg Q(x) \vee R(x)) \wedge P(a) \wedge Q(a) \wedge (\neg P(z) \vee \neg R(z))$
5. Forme clausale : $\{\neg Q(x) \vee R(x), P(a), Q(a), \neg P(z) \vee \neg R(z)\}$
6. Instances contradictoires : $\neg Q(a) \vee R(a), P(a), Q(a), \neg P(a) \vee \neg R(a)$

1 :	$\neg Q(a) \vee R(a)$	Hyp
2 :	$Q(a)$	Hyp
3 :	$R(a)$	Res 1,2
4 :	$\neg P(a) \vee \neg R(a)$	Hyp
5 :	$\neg P(a)$	Res 3,4
6 :	$P(a)$	Hyp
7 :	$\perp$	Res 5,6

Exercice 5, page 2

Instanciation : Nous obtenons les instances contradictoires suivantes :

$$\{R(a,a) + R(f(a),a), \quad R(a,a) + R(a,f(a)), \quad \overline{R(a,f(a))} + \overline{R(f(a),a)} + R(a,a), \quad \overline{R(a,a)}\}$$

Par résolution propositionnelle nous avons la preuve :

1 :	$R(a,a) + R(f(a),a)$	Hypothèse
2 :	$\overline{R(a,a)}$	Hypothèse
3 :	$R(f(a),a)$	Révolvant 1,2
4 :	$R(a,a) + R(a,f(a))$	Hypothèse
5 :	$R(a,f(a))$	Résolvant 2,4
6 :	$\overline{R(a,f(a))} + \overline{R(f(a),a)} + R(a,a)$	Hypothèse
7 :	$\overline{R(a,f(a))} + R(a,a)$	Résolvant 3,6
8 :	$R(a,a)$	Résolvant 5,7
9 :	$\perp$	Résolvant 8,2

Résolution binaire :

1 :	$R(x,y) + R(f(x),x)$	Hypothèse	
2 :	$\overline{R(y,y)}$	Hypothèse	
3 :	$\overline{R(t,t)}$	Copie 2	$< x := t >$
4 :	$R(f(t),t)$	Révolvant 1,3	$< x := t, y := t >$
5 :	$R(x,y) + R(x,f(x))$	Hypothèse	
6 :	$R(t,f(t))$	Résolvant 3,5	$< x := t, y := t >$
7 :	$\overline{R(x,y)} + \overline{R(y,z)} + R(x,z)$	Hypothèse	
8 :	$\overline{R(x,f(t))} + R(x,t)$	Résolvant 4,7	$< y := f(t), z := t >$
9 :	$R(u,f(u))$	Copie 6	$< t := u >$
10 :	$R(u,u)$	Résolvant 8,9	$< x := u, t := u >$
11 :	$\perp$	Résolvant 10,3	$< t := u >$

Exercice 6, page 2

1. $\forall x \forall y \forall z (x = y \wedge \neg(y = z) \Rightarrow \neg(x = z))$

contexte	numéro	preuve	règle
1	1	`Supposons` $x = y \wedge \neg(y = z)$	
1,2	2	`Supposons` $x = z$	
1,2	3	$x = y$	$\wedge E1$ 1
1,2	4	$y = z$	congruence 2, 3
1,2	5	$\neg(y = z)$	$\wedge E2$ 1
1,2	6	$\perp$	$\Rightarrow E$ 4, 5
1	7	`Donc` $\neg(x = z)$	$\Rightarrow I$ 2, 6
	8	`Donc` $x = y \wedge \neg(y = z) \Rightarrow \neg(x = z)$	$\Rightarrow I$ 1, 7
	9	$\forall z(x = y \wedge \neg(y = z) \Rightarrow \neg(x = z))$	$\forall I$ 8, z
	10	$\forall y \forall z(x = y \wedge \neg(y = z) \Rightarrow \neg(x = z))$	$\forall I$ 9, y
	11	$\forall x \forall y \forall z(x = y \wedge \neg(y = z) \Rightarrow \neg(x = z))$	$\forall I$ 10, x

2. $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$

contexte	numéro	preuve	règle
1	1	`Supposons` $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y)$	
1	2	$\exists x P(x)$	$\wedge E1$ 1
1,3	3	`Supposons` $P(x)$	
1,3,4	4	`Supposons` $P(y)$	
1,3,4	5	$\forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y)$	$\wedge E2$ 1
1,3,4	6	$\forall y (P(x) \wedge P(y) \Rightarrow x = y)$	$\forall E$ 5, x
1,3,4	7	$P(x) \wedge P(y) \Rightarrow x = y$	$\forall E$ 6, y
1,3,4	8	$P(x) \wedge P(y)$	$\wedge I$ 3, 4
1,3,4	9	$x = y$	$\Rightarrow E$ 8, 7
1,3	10	`Donc` $P(y) \Rightarrow x = y$	$\Rightarrow I$ 4, 9
1,3	11	$\forall y (P(y) \Rightarrow x = y)$	$\forall I$ 10, y
1,3	12	$P(x) \wedge \forall y (P(y) \Rightarrow x = y)$	$\wedge E$ 3, 11
1,3	13	$\exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$	$\exists I$ 12, x
1	14	`Donc` $P(x) \Rightarrow \exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$	$\Rightarrow I$ 3, 13
1	15	$\exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$	$\exists E$ 2, 14
	16	`Donc` $\exists x P(x) \wedge \forall x \forall y (P(x) \wedge P(y) \Rightarrow x = y) \Rightarrow \exists x (P(x) \wedge \forall y (P(y) \Rightarrow x = y))$	$\Rightarrow I$ 1,15

Exercice 7, page 2

1. $x \mid y = \bar{x}.\bar{y}$
2. Elle est commutative : $x \mid y \equiv \bar{x}.\bar{y} \equiv \bar{y}.\bar{x} \equiv y \mid x$
   Par contre $(1 \mid 0) \mid 0 = 0 \mid 0 = 1 \neq 0 = 1 \mid 1 = 1 \mid (0 \mid 0)$.
3. On vérifie que $x \mid x \equiv \bar{x}$.
4. $$\begin{aligned}
\top &\equiv \big(x \mid (x \mid x)\big) \mid \big(x \mid (x \mid x)\big) \\
\perp &\equiv x \mid (x \mid x) \\
\neg x &\equiv x \mid x \\
x \wedge y &\equiv (x \mid x) \mid (y \mid y) \\
x \vee y &\equiv (x \mid y) \mid (x \mid y) \\
x \Rightarrow y &\equiv \big((x \mid x) \mid y\big) \mid \big((x \mid x) \mid y\big)
\end{aligned}$$

BDDC v2
A basic bdd-based logical calculator

Pascal RAYMOND

November 24, 2008, (rev. September 30, 2022)

BDDC is a tool for manipulating logical formula. It is based on a Binary Decision Diagram library, that means that each formula is internally tranformed into a canonical form. In particular, if the formula (resp its negation) is a tautologie, BDDC will transform it into the "always true formula" (resp. the "always false formula").

All classical operators are provided. Moreover the tool provides a "functional-like" language that allows the user to define his (her) own logical operators.

1 Description

1.1 Basic features

The `bddc` utility implements an interactive processor for the evaluation of logical formula. It reads commands from the standard input, and prints the result to the standard output. The main commands are those which involve Boolean expressions: basically, `bddc` computes a bdd (binary decision diagram [1, 2]), which is a canonical representation of the expression.

For instance, if the user types the expression "`x + y.z ;`", the calculator interprets symbols `x`, `y` and `z` as Boolean arguments (i.e., basic propositions), and the whole expression as a Boolean function over those arguments. Note that "`+`" stands for logical *or*, and "`.`" operator for logical *and*. The calculator builds the bdd corresponding to the expression, and then prints a polynomial representation of this canonical form; in the example, it simply outputs "`x + y.z`" (useful, isn't it?).

If the expression is a tautology, its bdd is the always-true function: for instance, if the user types "`x => (y => x);`" (`=>` is the logical implication), the result is "`true`".

1.2 Advanced features

The user may assign an expression to a variable, and then use this variable to build a new expression ("`>>`" is the prompt symbol of the processor):

```
>>f := (y => x);
x + -y
>>x => f;
true
```

Moreover, the user may define new operators on bdds. For instance :

```
>>union(A, B) := A + B;
--new function: union, arity: 2
>>union(x.y, t);
x.y + t
```

Indeed, this example is quite stupid. Defining new operators becomes interesting when using control operators and recursion (see 5.3, 6).

2 Lexical aspects

Lists are coma-separated (`,`).

Identifiers are any strings starting with a letter, and containing only letters, digits and underscore symbol, excepting the following keywords:

```
and     constrain compare  cuts    cnf
del     dnf       else     exec    exist
false   forall    help     high    if
in      ite       implies  let     list
low     nor       not      or      print
quit    reed      restrict root    set
size    syntax    then     true    xor
```

The sequence `//` starts a comment, which ends at the next new-line. The sequence `/*` starts a comment, which ends at the next `*/` sequence.

3 Commands

The processor is started by the command `bddc`, then the user enters an interactive session.

quit

exits the processor.

help

prints the list of commands.

exec "*command-file*"

reads commands from a file.

syntax

prints the syntax of logical expressions.

list

prints the list of arguments and variables in the current context (see 5.2).

***exp*;**

builds the bdd corresponding to the expression in the current context, and outputs a polynomial equivalent to this bdd.

***ident* := *exp* ;**

builds the bdd corresponding to the expression in the current context, assigns it to the variable *ident*, and outputs a polynomial equivalent to this bdd.

***ident*(*identlist*) := *exp*;**

defines a function, wich can be used latter in logical expression.

print *ident*

print a polynomial equivalent to the bdd held by the variable *ident*. This command is obsolete: it is equivalent to type "*ident*;", since a identifier is a logical expression.

size *exp*;

prints the size of the bdd corresponding to *exp*. This size is expressed as a number of "bdd nodes" (see 5.1).

[dnf, cnf, reed] *exp*;

those commands are quite equivalent to typing "*exp*;", except that the output is printed (respectively) in disjunctive, conjunctive, or Reed-Muller normal form (see 5.5).

set [dnf, cnf, reed]

sets the output mode (disjunctive, conjunctive, or Reed-Muller normal form see 5.5).

4 Logical expressions

The calculator provides all the classical Boolean operators, plus some other specially dedicated to the bdd implementation. The syntax is infixed, and the expression can be typed on more than one line, so don't forget the terminating symbol "`;`"!

4.1 Boolean operators

false

(or "`0`") always-false function.

true

(or "`1`") always-true function.

ident

reference to an argument or a variable (depending on the current context).

not *exp*

(or "`-`*exp*") complement.

exp* or *exp

(or "*exp* `+` *exp*") disjunction.

exp* and *exp

(or "*exp* `.` *exp*") conjunction.

exp* xor *exp

(or "*exp* `<>` *exp*") exclusive or.

exp* => *exp

implication.

exp* = *exp

equivalence.

ite(*exp*,*exp*,*exp*)

functionnal "if then else"; "`ite(f,g,h)`" is equivalent to "`f.g + -f.h`".

forall *identlist exp*

universal quantifier.

exist *identlist exp*

existential quantifier.

`#` (*exp-list*)

at most one true in the list.

nor *exp-list*

none is true in the list.

xor (*exp-list*)

exactly one true in the list. Warning: xor (x1,x2,...,xi) is NOT "x1 xor x2 xor ... xor xi" (except for i=2).

4.2 Control operators

Those operators allow the user to control the way expressions are evaluated.

compare(*exp*,*exp*)

returns `1` (the always-true function) if the two expressions are logically equivalent, `0` (the always-false function) otherwise.

implies(*exp*,*exp*)

returns `1` if the first expression implies the second one, `0` otherwise.

cuts(*exp*,*exp*)

returns `1` if the intersection of the two expressions is not empty (i.e. their logical and is not `0`), `0` otherwise.

if *exp* then *exp* else *exp*

lazy "if": if the first operand is always-true (resp. always-false), the second (resp. third) operand is not evaluated; otherwise, it behaves as the `ite` operator.

let *ident* := *exp* in *exp*

local variable definition : the first expression is evaluated in a new context where *ident* is undefined. Then, the second expression is evaluated in a new context where the first result is associated to *ident*. The original context is then restaured, and the second result is returned (see 5.2 for details).

4.3 Bdd decomposition

The following operators are specific to the bdd representation: in particular, they do depend on the internal order of the arguments.

Every formula "`f`" that actually depends on at least one argument (i.e. neither `0` nor `1`) is equivalent to "`f = if root(f) then high(f) else low(f)`", where:

- `root(f)` is the least argument `f` actually depends on, according to the argument total order.
- neither `high(f)` nor `low(f)` depend on `root(f)`

See 5.4 for more details on Shannon decomposition and BDD.

root(*exp*)

the least argument which the expression depends on.

high(*exp*)

the "high" branch of the expression (i.e. the formula when the root-var is true).

low(*exp*)

the "low" branch of the expression (i.e. the formula when the root-var is false).

root(*exp-list*)

when called with more than one argument, the `root` function returns the minimal root-var of all the expressions. This generalisation is quite useful when defining recusive functions on the bdd stucture.

supp(*exp*)

returns the "support" of the exopression, that is, the "or" of all arguments the formula depends on.

4.4 Simplification modulo care-set

The following operators are specific to the bdd representation: in particular, they do depend on the internal order of the arguments.

It is often the case that a particular formula is manipulated under a set of assumptions, or "care-set".

restrict(*exp*,*exp*)

"`restrict(f,g)`" is a function equivalent to "`f`" if "`g`"; the result is simplier than "`f`" (less bdd nodes). The result is a function h satisfying:

$$if(f \wedge g = 0)\ \ then\ \ h = 0$$

$$if(f \vee \neg g = 1)\ \ then\ \ h = 1$$

Otherwise, h is some function satisfying:

$$(f \wedge g) \Rightarrow h \Rightarrow (f \vee \neg g)$$

N.B. The result is undefined if the second operand is `0`.

constrain(*exp*,*exp*)

(also called the generalized cofactor) is a kind of "restrict" operation. The result is not garanted to be "smaller" than f, but it has an interesting (but not trivial) property: let `h = constrain(f,g)`

$$h(x) = f(x')\ \text{ where }\ x' = \text{"the closer point to x in g"}$$

The precise definition of the "closer point ..." function is a little bit complicated (and depends on the variable ordering), but the main result is that the choice of x' **does not depend on** f.

N.B. The result is undefined if the second operand is `0`.

4.5 Priorities

Expressions can be enclosed within parenthesis. Some priorities are used to resolve ambiguities:

- `not`, `exist`, `forall`,
- `else`
- `and`
- `or`
- `=>` (right associative)

- `=`, `xor` (right associative)
- `in`

For instance:

```
let z := t = v in - exist x if y then x else z + z = y => z ;
```

is interpreted as:

```
let z := (t = v) in ((- (exist x (if y then x else z)) + z) = (y => z));
```

5 Appendix

5.1 Notes on BDD size

The tool uses a internal representation knowned as "signed binary decision diagrams" (SBDD). With this representation, a boolean function F and its negation $\neg F$ are (almost) represented by the same internal structure (they differ only on a boolean flag). With classic BDDs, the internal representation of F and $\neg F$ are disjoints. As a consequence, the cost of the negation is constant with SBDDs, since it is linear with classical BDDs. Moreover, it is known that this representation is always cheaper (in size and in time computation) than classical BDDs.

5.2 Notes on identifiers

The calculator deals with several kinds of identifiers:

arguments are the (global) free variables, i.e. identifiers that was once used without being first declared.

global variables are the identifiers introduced at the top level by a assignment command.

local variables are either formal parameters of a user-defined function, or identifiers introduced by a `let` operator.

The way an identifier is interpreted (i.e. the identifier's status) depends on the current context. Intuitively, at the top level, there is a global context, giving the status of arguments and global variables. The status of a global identifier is implicitly given by its first occurence: if it first appears in the left hand side of an assigment, the status is *variable*; if it first appears in an expression, the status is *argument*. One can type "`list`" to see the current global context :

```
>>x := a . b;
x := a . b
>>list
parameters are: a b
variables are: x
```

An argument cannot be re-used as a global variable; a variable, indeed, can be re-assigned:

```
>>a := x;
can't assign "a", not a variable
>>x := a + b;
x := a + b
```

Arguments are **globals**, while the scope of variables can be restricted using the `let ... in ...` operator.

```
>>y := let z := c + d in z;
y := c + d
>>list --note that z has disappeared, but not c and d:
arguments are: a b c d
variables are: x y
```

If `id` is a variable, `let id := ...` shadows the current value of `id`:

```
>>y;
c + d
>>z := let y := (let y := e.f in y.g) in x + y;
z := a + b + e.f.g
>>y;
c + d
```

Moreover, a local declaration shadows the status of the identifier:

```
>>a;
a
>>let a := e.f in a;
e.f
>>a;
a
```

5.3 Notes on user-defined functions

For the time being, no static verifications are made on user-defined functions. We can simply give the semantics for function calls: if the user has defined a function `foo(i1, ..., ik) := exp;`, then `foo(e1, ..., en)`, where `e1`, ..., `en` are correct expressions, is a correct expression equivalent to:

```
let i1 := e1 in
...
let in := en in
   exp;
```

In particular, a user-defined function may use free identifiers, but it is indeed quite dangerous: depending on the context call, those identifiers will be interpretted either as arguments, global variables, or even local variables of the current scope!

5.4 Notes on Shannon decomposition

The bdd decomposition is a special case of Shannon decomposition, where the choice of the decomposition variable is forced by the actual bdd representation. The general Shannon decomposition does not depend on the bdd structure, and can be obtained using logical operators; for all variable x and function f, we have $f = (x \wedge f1) \vee (\neg x \wedge f0)$, where (in bddc syntax):

```
f1 := exist x (x.f);  -- or better: f1 := constrain(f,x);
f0 := exist x (-x.f); -- or better: f0 := constrain(f,-x);
```

5.5 Notes on output format

By default, the tool prints the results in a disjunctive normal form (sum of "ands"). The idea is to make the result as "readable" as possible. The quality of the "result" dramaticaly depends on the internal order of the arguments. For example, depending on the order of its arguments, the expression "`if A then B else C`" can produce different (but equivalent) polynomials, such as "`A.B + -A.C + B.C`" or "`B.A + -A.C`". Indeed, the second form is "better" since it is more concise, but the problem of printing the simpliest polynomial equivalent to a given function is exponential. So, there is no guarantee on the "quality" of the print result.

The user can also prints the results in a conjonctive normal form (product of `or`'s). This is the dual of the first form.

The Reed-Muller normal form consists in expressing the boolean functions as polynomials in "and" and "xor" (the always-true function "1" is also necessary). This form is canonical (modulo the commutativity and associativity of the operators). For instance:

```
>>reed (if A then B else C);
>>B.A <> A.C <> C
>>reed (not A);
>>A <> 1
```

This form is interesting (since it is canonical), but it can seem "strange" and unreadable for most people! Moreover, there is no "simple" relation

between the size of the Reed-Muller form and a classical polynomial form: one form can be exponentially better or worse than the other:

```
>>A + B + C;
A + B + C

>>reed (A + B + C);
A.B.C <> A.B <> A.C <> A <> B.C <> B <> C

>>A <> B <> C;
A.B.C + A.-B.-C + -A.B.-C + -A.-B.C

>>reed (A <> B <> C);
A <> B <> C
```

6 Examples

6.1 Programming the negation

The following function implements the Boolean negation. Indeed, using this function is quite ineffective, since the built-in `not` operator does the same thing in constant time, but this simple algorithm shows the genaral structure of any unary bdd-based operator.

```
negation(a) :=
   if compare(a, true) then false
   else if compare(a, false) then true
   else ite( root(a), negation(high(a)), negation(low(a)));
```

In order to understand this algorithm, remain that any bdd `a` is either:

- `true`, i.e. the always true formula,
- `false`, i.e. the always false formula,
- a ternary "expression" of the form `ite(root(a), high(a), low(a))`, where `root(a)` is an argument [1], and `high(a)` and `low(a)` are two formulas not depending in `root(a)`.

The algorithm first treat the basic cases, by **comparing** the parameter to the constants `true` and `false`, and after the genaral case, where the parameter still depends on some argument. In this case, the function is recursively called on the corresponding high and low parts, and the results are combined under a test of the root argument.

[1] More precisely the identity formula associated to an argument

6.2 Finding an implicant in a bdd

This is another example of unary function, that builds, if it exists, an monomial that implies `a` :

```
path(a) :=
  if compare(a, true) then true
  else if compare(a, false) then false
  else if compare(high(a), false) then (not root(a) and path(low(a)))
  else (root(a) and path(high(a)))
;
```

6.3 Programming the union

The following function implements the Boolean or. The only interest is in showing how to build a binary operator. We first treat the basic cases, where either `a`, `b` are constants. Then, we treat the case where both `a` and `b` are "binary" bdds. In this last case, we use the binary extension of the `root` operator to get the "minimal" argument between `root(a)` and `root(b)`, and call it `min_root`. Then, the recursive calls are made by decomposing both parameters according to `min_root`.

```
union(a, b) :=
   if (compare(a, true) or compare(b, false)) then a
   else if (compare(b, true) or compare(a, false)) then b
   else let min_root := root(a, b) in
      ite(min_root,
          union(constrain(a, min_root), constrain(b, min_root)),
          union(constrain(a, not min_root), constrain(b, not min_root))
      );
```

The use of the generalized cofactor in the recursive calls may seem a little bit tricky. It is justified by the fact that (for instance) either `constrain(a, min_root) = a` if `min_root` $\neq$ `root(a)`, or `constrain(a, min_root) = high(a)` if `min_root` $=$ `root(a)`.

References

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6), 1978.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, 1986.

Natural Deduction

```
-p & -q => -(p + q)
```

```
assume -p & -q.
  assume p + q.
    assume p.
      -p.
      F.
    therefore p => F.
    assume q.
      -q.
      F.
    therefore q => F.
    F.
  therefore -(p + q).
therefore -p & -q => -(p + q).
```

Prove Formula | Annotate Proof | Annotate Proof for mobile | Go Back

examples | rules | syntax | info | download | home

Last Modified : 27-Feb-2023

Natural Deduction

```
(p & q => r) <=> (p => (q => r))
```

```
assume p & q => r.
  assume p.
    assume q.
      p & q.
      r.
    therefore q => r.
  therefore p => q => r.
therefore (p & q => r) => p => q => r.
assume p => q => r.
  assume p & q.
    p.
    q.
    q => r.
    r.
  therefore p & q => r.
therefore (p => q => r) => p & q => r.
p & q => r <=> p => q => r.
```

Prove Formula | Annotate Proof | Annotate Proof for mobile | Go Back

Natural Deduction

```
-p & -q => -(p + q)
```

```
assume -p & -q.
  assume p + q.
    assume p.
      -p.
      F.
    therefore p => F.
    assume q.
      -q.
      F.
    therefore q => F.
    F.
  therefore -(p + q).
therefore -p & -q => -(p + q).
```

Prove Formula | Annotate Proof | Annotate Proof for mobile | Go Back

examples | rules | syntax | info | download | home

Last Modified : 27-Feb-2023

Les notations des dérivées

Michel Lévy

31 mai 2023

Table des matières

1 Dérivées partielles des fonctions **2**

2 Dérivées des expressions **3**

3 Dérivées des dérivées **4**

4 Symétrie des dérivées de tous ordres **5**

5 Dérivation et composition des fonctions **7**

Références **8**

Résumé

On critique les notations usuelles des dérivées, car elles sont basées sur une confusion entre fonctions et expressions. On propose d'autres notations qui respectent la différence entre fonctions et expressions. Ma reflexion a été stimulée par la lecture du polycopié de Kevin Santugini. Il critique l'incohérence des notations des dérivées partielles mais renonce à en changer, car, écrit-il, "ces notations sont maintenant trop ancrées dans l'usage pour les remplacer". Je partage sa critique mais, contrairement à lui, je pense qu'il serait utile de changer ces anciennes notations, pour des notations plus conformes à l'usage des logiciels de calcul formel et aux essais de formaliser l'analyse dans les assistants de preuves.

1 Dérivées partielles des fonctions

Dès le début de l'article de wikipédia sur les dérivées partielles, la définition de ces dérivées est incorrecte. Je cite ce début : la dérivée partielle d'une fonction de plusieurs variables est sa dérivée par rapport à l'une de ses variables, les autres étant gardées constantes. Ce que je critique ici, c'est l'oubli de la différence entre les expressions, qui ont des variables, et les fonctions, qui n'ont pas de variables.

Dans la suite, je ne considère que des fonctions sur les réels. Une fonction à n arguments est une application de $D \to \mathbb{R}$ où $D \subset \mathbb{R}^n$. À mes yeux, l'expression "fonction de plusieurs variables" n'a aucun sens.

Soit f une fonction à deux arguments. Je note la dérivée *partielle* de f par rapport à son premier argument par $\frac{\partial f}{\partial 1}$. Cette dérivée partielle peut aussi être notée $\frac{\partial}{\partial 1}(f)$.

La dérivée partielle de f par rapport à son deuxième argument est notée $\frac{\partial f}{\partial 2}$. Cette dérivée partielle peut aussi être notée $\frac{\partial}{\partial 2}(f)$.

Ces deux dérivées sont aussi des fonctions à deux arguments et sont définies ainsi :

— pour tous $x, y \in \mathbb{R}$, $\frac{\partial f}{\partial 1}(x,y) = \lim\limits_{h \to 0} \frac{f(x+h,y)-f(x,y)}{h}$

— pour tous $x, y \in \mathbb{R}$, $\frac{\partial f}{\partial 2}(x,y) = \lim\limits_{h \to 0} \frac{f(x,y+h)-f(x,y)}{h}$

On étend ces notations aux fonctions à n arguments où n est un entier strictement positif. Soit f une telle fonction et i un entier tel que $1 \leq i \leq n$.

Pour tous $x_1, \ldots x_n \in \mathbb{R}$, $\frac{\partial f}{\partial i}(x_1, \ldots x_n) = \lim\limits_{h \to 0} \frac{f(x_1,\ldots x_{i-1},x_i+h,x_{i+1},\ldots x_n)-f(x_1,\ldots x_{i-1},x_i,x_{i+1},\ldots x_n)}{h}$

Lorsque f est une fonction à un seul argument, la fonction $\frac{\partial f}{\partial 1}$ à un seul argument est aussi notée f'.

Soit f une fonction à deux arguments. Puisqu'alors les fonctions $\frac{\partial f}{\partial 1}$ et de $\frac{\partial f}{\partial 2}$ sont comme f des fonctions à deux arguments, on peut définir les fonctions $\frac{\partial \frac{\partial f}{\partial 1}}{\partial 1}$, $\frac{\partial \frac{\partial f}{\partial 2}}{\partial 1}$, $\frac{\partial \frac{\partial f}{\partial 1}}{\partial 2}$, $\frac{\partial \frac{\partial f}{\partial 2}}{\partial 2}$, aussi à deux arguments, qui sont abrégées respectivement en $\frac{\partial f}{\partial 11}$, $\frac{\partial f}{\partial 12}$, $\frac{\partial f}{\partial 21}$, $\frac{\partial f}{\partial 22}$.

Notons par ε la liste vide. Par récurrence sur les listes u de 1 et 2, on peut définir $\frac{\partial f}{\partial u}$ ainsi :

— $\frac{\partial f}{\partial \varepsilon} = f$

— $\frac{\partial f}{\partial 1u} = \frac{\partial \frac{\partial f}{\partial u}}{\partial 1}$

— $\frac{\partial f}{\partial 2u} = \frac{\partial \frac{\partial f}{\partial u}}{\partial 2}$

On peut facilement généraliser ces notations pour des fonctions avec un nombre n d'arguments, en remplaçant ci-dessus les suites de 1 et 2 par des suites de $1, 2, \ldots n$.

Remarque 1 *Une liste d'éléments, dont chaque élément est représenté par un seul caractère, peut être représentée par la concaténation de ses éléments. Par exemple la liste des éléments* 1 2 3 2 *peut être représentée par* 1232*. Mais lorsque certains éléments sont représentés par plus d'un caractère, la liste est mise entre parenthèses et ses eléments sont séparés par une virgule. Par exemple la liste des éléments* 1 21 3 22 *est représentée par* $(1, 21, 3, 22)$*. Suivant le contexte, nous prendrons l'une ou l'autre représentation des listes.*

Pour un complément sur les fonctions de plusieurs variables, sur les notions de continuité et sur une présentation des dérivées partielles avec les notations classiques, je recommande la lecture du cours de Bernard Ycart.

2 Dérivées des expressions

Définition 2 *Expression*
Une expression est une constante, une variable ou une fonction appliquée à des expressions.

On suppose que l'ensemble des constantes et celui des variables sont disjoints. La syntaxe concrète des expressions n'est pas définie. Par exemple la somme des deux expressions $e1$ et $e2$ est notée par la syntaxe abstraite, dite aussi préfixée, $somme(e1, e2)$, la fonction somme est appliquée aux deux expressions $e1$ et $e2$, ou par la syntaxe concrète, appelée encore infixée, $e1 + e2$. De même le produit de ces deux expressions est notée par la syntaxe abstraite $produit(e1, e2)$ ou par l'une des trois syntaxes concrètes $e1 * e2$, $e1.e2$, $e1e2$.

Soit e une expression à valeur réelle et x une *variable* de type réel. La dérivée, de l'expression e par rapport à la variable x, est notée $\frac{de}{dx}$ et parfois aussi $\frac{d}{dx}(e)$.

Définition 3
Soit e une expression, x une variable et f la fonction définie par $f(x) = e$.
Soit f' la fonction dérivée de f. $\frac{de}{dx}$ est une expression *qui vérifie : $f'(x) = \frac{de}{dx}$.*
On introduit la notation $\frac{\delta e}{\delta x}$ comme désignant la dérivée f' de la fonction f.
Par suite $f'(x) = \frac{de}{dx} = \frac{\delta e}{\delta x}(x)$

Exemple 4 *Soit $f(x) = x^2$. Nous avons $f'(x) = \frac{dx^2}{dx} = 2 * x = \frac{\delta x^2}{\delta x}(x)$.*

Remarque 5 *Soit e une expression et x une variable. Dans le cours de Bernard Ycart, la notation $\frac{de}{dx}$ est utilisée avec le sens de $\frac{\delta e}{\delta x}$. Bernard Ycard a le mérite de garder toujours le même sens. Mais il est fréquent, dans beaucoup de cours et d'articles, d'utiliser la même notation $\frac{de}{dx}$ pour désigner l'expression $\frac{de}{dx}$ et la fonction $\frac{\delta e}{\delta x}$ et de faire confiance au contexte pour distinguer les deux sens.*

Dans la suite de ce document, on n'utilisera plus jamais la notation δ. On utilisera la notation $\frac{de}{dx}$ uniquement lorsque e est une *expression*.

Lemme 6
Soit f une fonction à deux arguments et x,y deux variables distinctes.
La première dérivée partielle de f est définie par : $\frac{\partial f}{\partial 1}(x,y) = \frac{d(f(x,y))}{dx}$
La dauxième dérivée partielle de f est définie par : $\frac{\partial f}{\partial 2}(x,y) = \frac{d(f(x,y))}{dy}$

Preuve : Soit g la fonction définie par $g(x) = f(x,y)$. D'après la définition 3, $\frac{d(g(x))}{dx} = g'(x)$. Par définition de g, $g'(x) = \lim_{h \to 0} \frac{f(x+h,y)-f(x,y)}{h}$. Par définition de la première dérivée partielle de f, et puisque $g(x) = f(x,y)$, nous avons donc :
$\frac{\partial f}{\partial 1}(x,y) = \frac{d(f(x,y))}{dx}$.
On prouve de façon analogue que $\frac{\partial f}{\partial 2}(x,y) = \frac{d(f(x,y))}{dy}$. □

On généralise le lemme ci-dessus aux fonctions ayant n arguments.

Lemme 7 *Soit f une fonction d'un domaine inclus dans $\mathbb{R}^n$ dans $\mathbb{R}$ et soient $x_1, \ldots x_n$ n variables distinctes.*
Nous avons pour tout i tel que $1 \leq i \leq n :$ $= \frac{\partial f}{\partial i}(x_1, \ldots x_n) = \frac{d(f(x_1, \ldots x_n))}{dx_i}$.

La preuve est omise car c'est une simple généralisation de la preuve du lemme précédent. Dans la notation traditionnelle, la dérivée partielle $\frac{\partial f}{\partial i}$ est abusivement notée $\frac{\partial f}{\partial x_i}$, en utilisant des variables arbitraires pour désigner les arguments de la fonction f.

3 Dérivées des dérivées

Puisque la dérivée d'une expression est une expression, on peut dériver la dérivée d'une expression.

Exemple 8 *dérivée de dérivée d'une expression*
$\frac{d(\frac{d(x*y^2+y*x^2)}{dx})}{dy} = \frac{d(y^2+y*2*x)}{dy} = 2*y+2*x$
On abrège $\frac{d(\frac{d(x*y^2+y*x^2)}{dx})}{dy}$ *en* $\frac{d(x*y^2+y*x^2)}{dyx}$

Soit e une expression et x , y deux variables. On abrège $\frac{d(\frac{de}{dy})}{dx}$ en $\frac{de}{dxy}$. Et plus généralement, on définit par récurrence la dérivée d'une expression par rapport à une liste de variables.

Définition 9 *dérivée d'une expression par rapport à une liste de variables*
Soit ε *la liste vide, soit x une variable, soit u une liste de variables, soit e une expression.*

1. $\frac{de}{d\varepsilon} = e$
2. $\frac{d\frac{de}{du}}{dx} = \frac{de}{dxu}$

On généralise le lemme 6 au cas de composition de plusieurs dérivations. On commence par le cas des fonctions à deux arguments.

Lemme 10 *Soit f une fonction à deux arguments. Nous avons :* $\frac{d(f(x,y))}{dxy} = \frac{\partial f}{\partial 12}(x,y)$, $\frac{d(f(x,y))}{dxx} = \frac{\partial f}{\partial 11}(x,y)$, $\frac{d(f(x,y))}{dyx} = \frac{\partial f}{\partial 21}(x,y)$, $\frac{d(f(x,y))}{dyy} = \frac{\partial f}{\partial 22}(x,y)$.

Preuve : Par définition des dérivées de dérivées des expressions, nous avons

$$\frac{d(f(x,y))}{dxy} = \frac{d(\frac{d(f(x,y))}{dy})}{dx}$$

D'après le lemme 6, nous avons

$$\frac{d(f(x,y))}{dy} = \frac{\partial f}{\partial 2}(x,y)$$

D'après ce même lemme, appliquée à l'expression $\frac{\partial f}{\partial 2}(x,y)$ (au lieu de l'expression $f(x,y)$) et à la fonction $\frac{\partial f}{\partial 2}$ (au lieu de la fonction f), nous avons

$$\frac{d(\frac{\partial f}{\partial 2}(x,y))}{dx} = \frac{\partial(\frac{\partial f}{\partial 2})}{\partial 1}(x,y)$$

Par définition des dérivées des fonctions (voir paragraphe 1) nous avons

$$\frac{\partial(\frac{\partial f}{\partial 2})}{\partial 1}(x,y) = \frac{\partial f}{\partial 12}(x,y)$$

D'après les quatre équations ci-dessus, nous avons $\frac{d(f(x,y))}{dxy} = \frac{\partial f}{\partial 12}(x,y)$. On prouve de façon analogue les autres égalités du lemme.

□

Nous généralisons le lemme ci-dessus, à un nombre quelconque d'arguments et de dérivations.

Lemme 11 *Soit f une fonction à n arguments. Pour tous les entiers $i_1, \ldots i_k$ compris entre 1 et n, nous avons :*

$$\frac{\partial f}{\partial i_1 \ldots i_k}(x_1, \ldots x_n) = \frac{d(f(x_1, \ldots x_n))}{dx_{i_1} \ldots x_{i_k}}$$

Nous ne faisons pas la preuve de ce lemme, car c'est une simple généralisation du lemme précédent. La dérivée $\frac{d(f(x_1,\ldots x_n))}{dx_{i_1}\ldots x_{i_k}}$ de l'expression $f(x_1, \ldots x_n)$ est une dérivée d'ordre k de l'expression. De même la dérivée $\frac{\partial f}{\partial i_1 \ldots i_k}$ de la fonction f est aussi appelée dérivée d'ordre k de la fonction.

4 Symétrie des dérivées de tous ordres

Soit f une fonction sur un domaine inclus dans $\mathbb{R}^n$ dans $\mathbb{R}$. Il avait été affirmé par Bernouilli, Clairault, Young, avec des preuves incorrectes, ainsi qu'il est constaté dans

la version anglaise de wikipedia que pour tous i, j où $1 \leq i, j \leq n$,

$$\frac{\partial f}{\partial ij} = \frac{\partial f}{\partial ji} \tag{1}$$

Attention, j'emploie ici la notation des dérivées partielles définie au paragraphe 1. Dans la version usuelle, la dernière égalité est notée $\frac{\partial f}{\partial x_i x_j} = \frac{\partial f}{\partial x_j x_i}$. Les variables sont utilisées (à tord à mon avis) pour désigner les arguments de la fonction f, ce qui crée une confusion entre les expressions qu'on dérive relativement à une variable et les fonctions qu'on dérive relativement à un de leurs arguments.

Le theorème de Schwarz donne des conditions suffisantes pour que l'égalité 1 soit vérifiée.

Théorème 12 *(Théorème de Schwarz)*
Soit f une fonction sur les réels définie sur un ouvert D où $D \subset \mathbb{R}^n$. Soient i, j deux entiers tels que $1 \leq i < j \leq n$. Supposons que les deux dérivées partielles $\frac{\partial f}{\partial ij}$ et $\frac{\partial f}{\partial ji}$ sont continues sur D. Alors, pour tout $a \in D$, $\frac{\partial f}{\partial ij}(a) = \frac{\partial f}{\partial ji}(a)$

La preuve de ce théorème est (très bien) faite dans l'article du wikipedia anglais référencée ci-dessus. Puisqu'une permutation d'une liste d'entiers est une combinaison de transpositions élémentaires, le théorème de Schwarz peut être généralisé aux dérivations de tous ordres.

Définition 13 *Fonction de classe k (où $k \geq 0$) sur un domaine*
Une fonction f définie sur domaine ouvert D où $D \subset \mathbb{R}^n$ est de classe k sur ce domaine, si elle et ses dérivées partielles d'ordre au plus k sont continues sur ce domaine.

Théorème 14 *(Théorème de Schwarz généralisé)*
Soit f une fonction sur les réels de classe k sur le domaine ouvert D où $D \subset \mathbb{R}^n$. Si u et v sont des listes de longueur p, où $p \leq k$ d'entiers entre 1 et n et si ces listes sont des permutations l'une de l'autre, alors pour tout $a \in D$, $\frac{\partial f}{\partial u}(a) = \frac{\partial f}{\partial v}(a)$

Preuve : Une transposition élémentaire est un échange entre deux éléments consécutifs d'une liste. Puisque toute permutation est un produit de transpositions élémentaires, il suffit de vérifier le théorème lorsque v est obtenue par une transposition élémentaire effectuée sur u, autrement dit lorsqu'il existe deux listes w et x, deux entiers i et j tels que $u = wijx$ et $v = wjix$.

Soit $a \in D$. Par définition des dérivées partielles des fonctions, nous avons

$$\frac{\partial f}{\partial wijx}(a) = \frac{\partial(\frac{\partial f}{\partial ijx})}{\partial w}(a) \tag{2}$$

$$= \frac{\partial(\frac{\partial(\frac{\partial f}{\partial x})}{\partial ij})}{\partial w}(a) \tag{3}$$

Puisque la fonction f est de classe k sur D, la fonction dérivée $\frac{\partial f}{\partial x}$ est continue et deux fois dérivable sur ce domaine. Par suite on peut lui appliquer le théorème de Schwartz et par conséquent pour tout $a \in D$

$$\frac{\partial(\frac{\partial f}{\partial x})}{\partial ij}(a) = \frac{\partial(\frac{\partial f}{\partial x})}{\partial ji}(a) \tag{4}$$

D'après la définition des dérivées partielles des fonctions, il en résulte que pour tout $a \in D$ nous avons

$$\frac{\partial f}{\partial ijx}(a) = \frac{\partial f}{\partial jix}(a) \tag{5}$$

En dérivant par w, on en déduit que pour tout $a \in D$, $\frac{\partial f}{\partial wijx}(a) = \frac{\partial f}{\partial wjix}(a)$, ce qu'il fallait prouver pour la transposition élémentaire de ij en ji.

□

Le logiciel xcas, voir [1] et [2], écrit et maintenu par Bernard Parisse, est un logiciel de calcul formel, avec d'innombrables possibilités. J'indique ici comment dériver une expression relativement à des variables.

Le calcul de $\frac{d(x^3 * y^2)}{dyxx}$ est réalisé par la commande $diff(x\text{^}3 * y\text{^}2, y, x, x)$.

Le corollaire suivant montre que l'ordre des dérivations est sans importance, c'est-à_dire que $diff(x\text{^}3 * y\text{^}2, y, x, x) = diff(x\text{^}3 * y\text{^}2, x, y, x) = diff(x\text{^}3 * y\text{^}2, x, x, y)$.

Cela justifie la notations par xcas de cette dérivation par la commande $diff(x\text{^}3 * y\text{^}2, x\$2, y\$1)$, les indications $x\$2$ et $y\$1$ précisent qu'il faut faire deux dérivations selon x et une dérivation selon y, l'ordre de ces dérivations étant sans importance.

Corollaire 15 *(Théorème de Schwartz et dérivées des expressions)*
Soit D un domaine ouvert inclus dans $\mathbb{R}^n$ et soit f une fonction de classe k (où $k \geq 2$) sur ce domaine. Soit p un entier où $p \leq k$, soit $i_1, \ldots i_p$ une liste d'entiers entre 1 et n. Nous avons

$$\frac{d(f(x_1, \ldots x_n))}{dx_{i_1} \ldots x_{i_p}} = \frac{d(f(x_1, \ldots x_n))}{dx_{j_1} \ldots x_{j_p}} \tag{6}$$

où $j_1, \ldots j_p$ est une permutation quelconque de $i_1, \ldots i_p$.

Preuve : D'après le lemme 11, nous avons $\frac{d(f(x_1, \ldots x_n))}{dx_{i_1} \ldots x_{i_k}} = \frac{\partial f}{\partial i_1 \ldots i_k}(x_1, \ldots x_n)$.

D'après le théorème de Schwartz généralisé , nous avons $\frac{\partial f}{\partial i_1 \ldots i_k}(x_1, \ldots x_n) = \frac{\partial f}{\partial j_1 \ldots j_k}(x_1, \ldots x_n)$ D'après le lemme 11, nous avons $\frac{\partial f}{\partial j_1 \ldots j_k}(x_1, \ldots x_n) = \frac{d(f(x_1, \ldots x_n))}{dx_{j_1} \ldots x_{j_p}}$ D'après ces trois égalités, on a $\frac{d(f(x_1, \ldots x_n))}{dx_{i_1} \ldots x_{i_p}} = \frac{d(f(x_1, \ldots x_n))}{dx_{j_1} \ldots x_{j_p}}$ □

5 Dérivation et composition des fonctions

Théorème 16 *Soient $e_1, \ldots e_n$ une suite d'expressions. Soit x une variable. Soit f une fonction sur les réels définie sur un domaine D inclus dans $\mathbb{R}^n$. Supposons que f est de classe C_1 sur le domaine D et que $(e_1, \ldots e_n)$ est dans ce domaine. Supposons que pour*

tout i et pour x les dérivées $\frac{d(e_i)}{dx}$ *sont définies. Lorsque ces conditions sont vérifiées, nous avons :*

$$\frac{d(f(e_1,\dots e_n))}{dx} = \sum_{i=1}^{n}(\frac{\partial f}{\partial i})(e_1,\dots e_n) * \frac{d(e_i)}{dx}$$

Ce théorème est souvent énoncé comme une relation entre différentielles :

$$d(f(e_1,\dots e_n)) = \sum_{i=1}^{n}(\frac{\partial f}{\partial i})(e_1,\dots e_n) * d(e_i)$$

En divisant par dx, on obtient le théorème ci-dessus.
Le théorème reste vrai lorsque f au lieu d'être une fonction d'un domaine D où $D \subset \mathbb{R}^n$ dans $\mathbb{R}$ est une fonction de D dans $\mathbb{R}^p$.

Exemple 17 *Vérifions le théorème 16 sur un exemple. Soit f définie par $f(x,y) = 2xy$. Calculons de deux manières distinctes* $\frac{d(f(2x,3y))}{dx}$.

Par définition de f, $\frac{d(f(2x,3y))}{dx} = \frac{d(12xy)}{dx} = 12y$.

D'après le lemme 6, nous avons

— $\frac{\partial f}{\partial 1}(x,y) = \frac{d(f(x,y))}{dx} = \frac{d(2xy)}{dx} = 2y$

— $\frac{\partial f}{\partial 2}(x,y) = \frac{d(f(x,y))}{dy} = \frac{d(2xy)}{dy} = 2x$

Calculons $\frac{\partial f}{\partial 1}(2x,3y) * \frac{d(2x)}{dx} + \frac{\partial f}{\partial 2}(2x,3y) * \frac{d(3y)}{dx}$.

L'expression $\frac{d(3y)}{dx}$ *valant* 0, $\frac{\partial f}{\partial 1}(2x,3y)$ *valant* $6y$, $\frac{d(2x)}{dx}$ *valant* 2, *la somme ci-dessus vaut* $12y$, *ce qui est bien la valeur de* $\frac{d(f(2x,3y))}{dx}$.

Lorsque la fonction f n'a qu'un argument, le théorème 16 implique le corollaire suivant qui, en français, est le théorème de la dérivée des fonctions composées et, en anglais, est la Chain rule.

Corollaire 18 *Soit f et g deux fonctions à un argument. Si g est dérivable en x et f est dérivable en $g(x)$ alors* $(f \circ g)'(x) = f'(g(x)) * g'(x)$.

Preuve : Posons $g(x) = e$. Puisque g est dérivable en x et que $g'(x) = \frac{d(e)}{dx}$, l'expression $\frac{d(e)}{dx}$ est définie. Puisque f est dérivable en $g(x)$ et que $f'(g(x)) = \frac{d(f(e))}{dx}$, l'expression $\frac{d(f(e))}{dx}$ est aussi définie. Donc, par application de théorème 16 dans le cas où $n = 1$, nous avons $\frac{d(f(e))}{dx} = f'(e) * \frac{de}{dx}$. En remplaçant e par $g(x)$, nous obtenons
$\frac{d(f(g(x)))}{dx} = \frac{d(f \circ g)(x)}{dx} = (f \circ g)'(x) = f'(g(x)) * \frac{d(g(x))}{dx} = f'(g(x)) * g'(x)$. □

Références

[1] Bernard PARISSE. *xcas doc*. URL : https://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html (visité le 27/11/2021).

[2] Bernard PARISSE. *xcas en ligne*. URL : https://www-fourier.ujf-grenoble.fr/~parisse/xcasfr.html (visité le 19/11/2021).

[3] Kevin SANTUGINI-REPIQUET. *PolyDeriveesPartielles*. URL : https://www.math.u-bordeaux.fr/~ksantugi/downloads/PolyDeriveesPartielles.pdf (visité le 30/11/2021).

[4] Bernard YCART. *Fonctions de plusieurs variables*. URL : https://membres-ljk.imag.fr/Bernard.Ycart/mel/fp/ (visité le 19/11/2021).

Bijection entre les topologies d'Alexandroff et les préordres

Michel Lévy

21 janvier 2015

Définition 1 (Préordre)

Un préordre est un couple (E,R) où R est une relation réflexive et transitive sur l'ensemble E.

Soient $e,f \in E$ tel que eRf : f est un successeur de e.

*Soit A un sous-ensemble de E. L'*origine *de A, relativement au préordre, est l'ensemble des élements de E dont tous les successeurs sont éléments de A. L'origine de A est notée $o_{(E,R)}(A)$, et souvent $o(A)$ quand le préordre est implicite.*

Lemme 2 (Propriété de l'origine d'un ensemble)

Soit (E,R) un préordre. L'origine vérifie les propriétés suivantes :

1. $o(\emptyset) = \emptyset$
2. $o(E) = E$
3. *Pour tous $A,B \subset E$, si $A \subset B$ alors $o(A) \subset o(B)$*
4. *Pour tout $A \subset E$, $o(A) \subset A$*
5. *Pour tout $A \subset E$, $o(o(A)) = o(A)$*

Preuve : Les propriétés 1, 2, 3 sont des conséquences immédiates de la définition de l'origine.

Soit $a \in o(A)$. Tout successeur de a est dans A. Puisque R est reflexive, a est son propre successeur, donc $a \in A$, ce qui prouve 4.

Prouvons la propriété 5. D'après la propriété précédente, $o(o(A)) \subset o(A)$. Il suffit donc de prouver que $o(A) \subset o(o(A))$.

Soit $a \in o(A)$. Tous les successeurs de a sont éléments de A. Soit b un successeur de a. Tout successeur de b est, par transitivité de R, aussi un successeur de a donc un élément de A. Par suite $b \in o(A)$. Donc tous les successeurs de a sont dans $o(A)$. Et par conséquent $a \in o(o(A))$

□

Définition 3 (Topologie)

Une topologie est un couple (E,T) où T est un ensemble de sous-ensembles de E, contenant E et $\emptyset$, tel que l'intersection de deux éléments de T est un élément de T et l'union d'éléments de T est un élément de T. Un élément de T est un ouvert *de la topologie.*

Une topologie d'Alexandroff[1] *une topologie* (E,T) *où l'intersection d'éléments de* T *est un élément de* T.

Définition 4 (Topologie associée à un préordre) *Soit* (E,R) *un préordre. Soit* T *l'ensemble des* $A \subset E$ *tels que* $o(A) = A$. (E,T) *est la topologie d'Alexandroff associée au préordre.*

Preuve : D'après les propriétés de o, l'ensemble vide et E sont des éléments de T.

Montrons que l'union d'un ensemble d'éléments de T est aussi un élément de T. Soit S un sous-ensemble de T. Montrons que $o(\bigcup S) = \bigcup S$. D'après les propriétés de o, on sait que $o(\bigcup S) \subset \bigcup S$. Il suffit donc de montrer l'inclusion inverse. Soit $a \in \bigcup S$. Il existe $A \in S$ tel que $a \in A$. Puisque A est élément de T, $o(A) = A$, donc tous les successeurs de a sont dans A, donc dans $\bigcup S$. Par suite $a \in o(\bigcup S)$.

Montrons que l'intersection d'un ensemble d'éléments de T est aussi un élément de T. Soit S un sous-ensemble de T. Montrons que $o(\bigcap S) = \bigcap S$. Comme ci-dessus, il suffit de montrer que $\bigcap S \subset o(\bigcap S)$. Soit $a \in \bigcap S$. Pour tout $A \in S$, $a \in A$. Puisque S est sous-ensemble de T, pour tout $A \in S$, $a \in o(A)$. Donc pour tout $A \in S$, pour tout b successeur de a, on a $b \in A$. Par suite pour tout b successeur de a, pour tout $A \in S$, $b \in A$. Par conséquent tous les successeurs de a sont éléments de $\bigcap S$, donc $a \in o(\bigcap S)$.

Par suite, (E,T) est une topologie d'Alexandroff.

□

Définition 5 (Préordre associé à une topologie) *Soit* (E,T) *une topologie. Soit* R *la relation suivante sur* E *:* eRf *si et seulement si* $\forall A \in T(e \in A \Rightarrow f \in A)$. *Par définition de* R*, il est évident que* (E,R) *est un préordre, le préordre* associé *à la topologie.*

Théorème 6

Soit (E,T) *une topologie. Soit* (E,R) *le préordre associée à cette topologie.*

Soit (E,T') *la topologie associée au préordre* (E,R). *On rappelle que* T' *est l'ensemble des* $A \subset E$ *tels que* $o(A) = A$.

$T \subset T'$*, comme on dit usuellement* (E,T') *est une topologie plus fine que* (E,T).

Si T *est une topologie d'Alexandroff,* $T = T'$.

Preuve : Montrons que $T \subset T'$.

Soit $A \in T$ et $a \in A$. Soit b tel que aRb, un successeur de a. Puisque (E,R) est le préordre associé à la topologie (E,T), $\forall B \in T(a \in B \Rightarrow b \in B)$. Puisque $A \in T$ et $a \in A$, nous avons $b \in A$. Donc, puisque tous les successeurs de a sont dans A, $a \in o(A)$. Par conséquent $A \subset o(A)$. Puisque, d'après les propriétés de o (cf 2, $o(A) \subset A$, on a $A = o(A)$, donc $A \in T'$.

Supposons que (E,T) est une topologie d'Alexandroff, autrement dit l'intersection d'éléments de T est un élément de T. Montrons que $T' \subset T$.

Soit $A \in T'$. Puisque $o(A) = A$, si $a \in A$ et b successeur de a alors $b \in A$. Autrement dit, par définition de R : $a \in A \wedge \forall B \in T(a \in B \Rightarrow b \in B) \Rightarrow b \in A$.

Puisque (E,T) est une topologie d'Alexandroff, pour $a \in A$, le plus petit ouvert contenant a existe, car c'est l'intersection des ouverts (autrement dit, des éléments de T) contenant a. Notons $ppo(a)$ cet ouvert.

1. https://fr.wikipedia.org/wiki/Topologie_d'Alexandroff)

Soit $D = \bigcup\{ppo(a) \mid a \in A\}$. Puisque D est une union d'ouverts, c'est un ouvert, c'est-à-dire un élément de T. Par définition de D, $A \subset D$.

Montrons que $D \subset A$. Soit $b \in D$. Il existe $a \in A$ tel que $b \in ppo(A)$. La condition $b \in ppo(a)$ s'écrit encore $\forall B \in T(a \in B \Rightarrow b \in B)$.

Puisque A vérifie $a \in A \wedge \forall B \in T(a \in B \Rightarrow b \in B) \Rightarrow b \in A$, il en résulte que $b \in A$.

Par suite $A = D$ donc $A \in T$.

□

Théorème 7 *La correspondance entre une topologie d'Alexandroff (E,T) et son préordre associé (E,R) où eRf si et seulement si $\forall A \in T(e \in A \Rightarrow f \in B)$ est une bijection.*

Preuve : D'après le théorème 6, cette correspondance est une injection. Puisque la topologie associée à un préordre, est une topologie d'Alexandroff, cette correspondance est surjective. □

Sens de Kripke et de Tarski de la logique intuitionniste

Michel Lévy

16 janvier 2015

La logique intuitionniste (propositionnelle) est présentée avec deux sémantiques en apparence très différente : la sémantique de Kripke et celle de Tarski. Nous donnons la définition de ces deux sémantiques, puis nous montrons comment transformer un modèle de Kripke en un modèle de Tarski et réciproquement.

L'ensemble des formules de la logique intuitionniste est l'ensemble construit à partir d'un ensemble de variables (noté *Var*) et des connectives $\wedge, \vee, \Rightarrow, \perp$. Une variable est une suite de minuscules avec ou sans indice. Les majuscules dénotent des formules. La négation et l'équivalence sont considérées comme des abréviations

1. $\neg A = A \Rightarrow \perp$
2. $A \Leftrightarrow B = (A \Rightarrow B) \wedge (B \Rightarrow A)$

$Formule(Var)$ est l'ensemble des formules dont les variables sont éléments de *Var*. Dans la suite toutes les formules sont implicitement éléments de cet ensemble et les variables sont éléments de *Var*.

1 Définition des deux sémantiques

1.1 Sémantique de Kripke

Définition 1.1 (Préordre - Interprétation de Kripke)

Un préordre est un couple (E,R) où R est une relation réflexive et transitive sur l'ensemble E.

Soient $e, f \in E$ tel que eRf. L'élément f est un successeur de e.

*Soit A un sous-ensemble de E. L'*origine *de A, relativement au préordre, est l'ensemble des élements de E dont tous les successeurs sont éléments de A. L'origine de A est notée $o_{(E,R)}(A)$, et souvent $o(A)$ quand le préordre est implicite.*

Une interprétation de Kripke *est un triplet (E,R,v) où (E,R) est un préordre et $v : Var \mapsto 2^E$ est une application qui vérifie la propriété suivante : pour tout $x \in Var$, si $e \in v(x)$ et $e\ R\ f$ alors $f \in v(x)$.*

Définition 1.2 (Valeur de Kripke d'une formule)

Soit $I = (E,R,v)$ une interprétation de Kripke. On étend l'application v en une application $v_K^I : Formule(Var) \mapsto 2^E$ de la façon suivante. Soient x une variable et A, B deux formules :

1. $v_K^I(x) = v(x)$
2. $v_k^I(\perp) = \emptyset$
3. $v_K^I(A \vee B) = v_K^I(A) \cup v_K^I(B)$
4. $v_K^I(A \wedge B) = v_K^I(A) \cap v_K^I(B)$
5. $v_K^I(A \Rightarrow B) = o_{(E,R)}(\overline{v_K^I(A)} \cup v_K^I(B)\}$

Définition 1.3 (Formule Kripke-valide)

La formule A est Kripke-valide si et seulement si pour tout interprétation de Kripke $I = (E,R,v)$, $v_K^I(A) = E$.

Dans le document http://teachinglogic.liglab.fr/INT1/S4Lint.pdf, on trouve une autre présentation de la sémantique de Kripke et nous expliquons ici comment correspondent ces deux présentations.

Définition 1.4 (Assignation)

Soit $I = (E,R,v)$ une interprétation de Kripke. Une assignation est un couple Ie composé de l'interprétation I et d'un élement $e \in E$ de cette interprétation. Une assignation permet de donner une valeur de vérité à une formule.

Dans le document cité, on trouvera une définition de la phrase $Ie \models A$, l'assignation Ie est *rend vraie* la formule A. Il est clair que ces deux définitions de la sémantique de Kripke sont presque identiques, car elles vérifient : $\{e \in E \mid Ie \models A\} = v_K^I(A)$.

Autrement dit, relativement à une interprétation, la valeur d'une formule est l'ensemble des éléments rendant vraie la formule.

Une formule est Kripke-valide si et seulement si toute assignation rend vraie la formule.

1.2 Sémantique de Tarski

Définition 1.5 (Topologie d'Alexandroff - Interprétation de Tarski)

Une topologie est un couple (E,T) où T est un ensemble de sous-ensembles de E, contenant E et $\emptyset$, tel que l'intersection de deux éléments de T est un élément de T et l'union d'éléments de T est un élément de T. Un élément de T est un ouvert *de la topologie.*

Soit A un sous-ensemble de E. L'intérieur de A, relativement à la topologie, est le plus grand ouvert inclus dans A. L'intérieur de A est noté $i_{(E,T)}(A)$ et souvent $i(A)$ quand la topologie est implicite.

Une topologie d'Alexandroff (cf wikipédia https://fr.wikipedia.org/wiki/Topologie_d'Alexandroff) est une topologie (E,T) où l'intersection d'éléments de T est un élément de T.

Une interprétation de Tarski *est un triplet (E,T,v) où (E,T) est une topologie d'Alexandroff et $v : Var \mapsto 2^E$ est une application qui vérifie la propriété suivante : pour tout $x \in Var$, $v(x)$ est un ouvert.*

Définition 1.6 (Valeur de Tarski d'une formule)

Soit $I = (E,T,v)$ une interprétation de Tarski. On étend l'application v en une application $v_T^I : Formule(Var) \mapsto 2^E$ de la façon suivante. Soient x une variable et A,B deux formules :

1. $v_T^I(x) = v(x)$
2. $v_T^I(\bot) = \emptyset$
3. $v_T^I(A \vee B) = v_T^I(A) \cup v_T^I(B)$
4. $v_T^I(A \wedge B) = v_T^I(A) \cap v_T^I(B)$
5. $v_T^I(A \Rightarrow B) = i_{(E,T)}(\overline{v_T^I(A)} \cup v_T^I(B))$

Il est évident (par récurrence sur les formules), que la valeur d'une formule est un ouvert.

Définition 1.7 (Formule Tarski-valide) *Une formule A est Tarski-valide si et seulement si pour tout interprétation $I = (E,T,v)$ de Tarski, $v_T^I(A) = E$.*

2 Relation entre les sémantiques de Kripke et de Tarski

2.1 Préordre et espace topologique associé

Lemme 2.1 (Propriété de l'origine d'un ensemble)

Soit (E,R) un préordre. Soit A un sous-ensemble de E, on rappelle que l'origine de A, notée $o(A)$ est l'ensemble des éléments de E dont tous les successeurs sont éléments de A. L'origine vérifie les propriétés suivantes :

1. $o(\emptyset) = \emptyset$
2. $o(E) = E$
3. *Pour tous $A,B \subset E$, si $A \subset B$ alors $o(A) \subset o(B)$*
4. *Pour tout $A \subset E$, $o(A) \subset A$*
5. *Pour tout $A \subset E$, $o(o(A)) = o(A)$*

Preuve : Les propriétés 1, 2, 3 sont des conséquences immédiates de la définition de l'origine.

Soit $a \in o(A)$. Tout successeur de a est dans A. Puisque R est reflexive, a est son propre successeur, donc $a \in A$, ce qui prouve 4.

Prouvons la propriété 5. D'après la propriété précédente, $o(o(A)) \subset o(A)$. Il suffit donc de prouver que $o(A) \subset o(o(A))$.

Soit $a \in o(A)$. Tous les successeurs de a sont éléments de A. Soit b un successeur de a. Tout successeur de b est, par transitivité de R, aussi un successeur de a donc un élément de A. Par suite $b \in o(A)$. Donc tous les successeurs de a sont dans $o(A)$. Et par conséquent $a \in o(o(A))$

□

Lemme 2.2 (Préordre et topologie associée)

Soit (E,R) un préordre.

Soit T l'ensemble des sous-ensembles $A \subset E$ tel que $o(A) = A$. (E,T) est la topologie d'Alexandroff associée au préordre.

La topologie associée au préordre *vérifie que pour tout $A \subset E$, $o(A) = i(A)$.*

Preuve : D'après les propriétés de o, l'ensemble vide et E sont des éléments de T.

Montrons que l'union d'un ensemble d'éléments de T est aussi un élément de T. Soit S un sous-ensemble de T. Montrons que $o(\bigcup S) = \bigcup S$. D'après les propriétés de o, on sait que $o(\bigcup S) \subset \bigcup S$. Il suffit donc de montrer l'inclusion inverse. Soit $a \in \bigcup S$. Il existe $A \in S$ tel que $a \in A$. Puisque A est élément de T, $o(A) = A$, donc tous les successeurs de a sont dans A, donc dans $\bigcup S$. Par suite $a \in o(\bigcup S)$.

Montrons que l'intersection d'un ensemble d'éléments de T est aussi un élément de T. Soit S un sous-ensemble de T. Montrons que $o(\bigcap S) = \bigcap S$. Comme ci-dessus, il suffit de montrer que $\bigcap S \subset o(\bigcap S)$. Soit $a \in \bigcap S$. Pour tout $A \in S$, $a \in A$. Puisque S est sous-ensemble de T, pour tout $A \in S$, $a \in o(A)$. Donc pour tout $A \in S$, pour tout b successeur de a, on a $b \in A$. Par suite pour tout b successeur de a, pour tout $A \in S$, $b \in A$. Par conséquent tous les successeurs de a sont éléments de $\bigcap S$, donc $a \in o(\bigcap S)$.

Nous venons de prouver que (E,T) est une topologie d'Alexandroff.

Puisque (E,T) est une topologie, les éléments de T sont appelés des ouverts. Soit A un sous-ensemble de E. On rappelle que $o(A)$, l'origine de A est l'ensemble des éléments de E dont tous les successeurs sont dans A et que $i(A)$, l'intérieur de A est le plus grand ouvert inclus dans A. Montrons que l'intérieur et l'origine d'un ensemble sont identiques.

D'après les propriétés de o, $o(o(A)) = o(A)$ et $o(A) \subset A$, donc $o(A)$ est un ouvert inclus dans A. Par suite $o(A) \subset i(A)$.

Supposons que $o(A)$ n'est pas égal à $i(A)$, donc n'est pas le plus grand ouvert inclus dans A. Il existerait un ouvert B tel que $o(A) \subset B \subset A$ et $B \neq o(A)$. Puisque $B \subset A$, par monotonie de o, $o(B) \subset o(A)$. Puisque B est un ouvert, $o(B) = B$ donc $B \subset o(A)$ et par suite $B = o(A)$. Par suite $o(A) = i(A)$. □

Lemme 2.3 (De Kripke vers Tarski)

Soit $I = (E,R,v)$ une interprétation de Kripke.

Soit (E,T) la topologie associée au préordre (E,R) et $J = (E,T,v)$ l'interprétation de Tarski associée *à I. Pour toute formule A, $v_K^I(A) = v_T^J(A)$.*

Preuve : Le lemme est prouvé par récurrence sur l'ensemble des formules. Il est la conséquence immédiate de ce que, d'après 2.2, pour tout $A \subset E$, on a $o(A) = i(A)$ où $o(A)$ est l'origine de A pour le préordre (E,R) et $i(A)$ est l'intérieur de A pour la topologie (E,T) *associée* au préordre. □

2.2 Espace topologique et préordre associée

Définition 2.4 (Topologie et préordre associée)

Soit (E,T) une topologie. Soit R la relation sur E ainsi définie : soient $e, f \in E$, eRf si et seulement pour tout ouvert $A \in T$, si $e \in A$ alors $f \in A$.

D'après cette définition, il est évident R est une relation réflexive et transitive donc que (E,R) est un préordre, le préordre associé à la topologie.

Lemme 2.5

Soit (E,T) une topologie d'Alexandroff et (E,R) le préordre qui lui est associé. Pour tout A sous-ensemble de E, on a $o(A) = i(A)$.

Preuve : Soit (E,T) une topologie d'Alexandroff et (E,R) le préordre qui lui est associé.

Soit (E,T') la topologie associée à ce préordre. Nous avons vu dans le lemme 2.2 que (E,T') est une topologie d'Alexandroff et que pour tout $A \subset E$, $o_{(E,R)}(A) = i_{(E,T')}(A)$.

Dans l'article de wikipédia `https://fr.wikipedia.org/wiki/Topologie_d'Alexandroff`, il est indiqué, que si (E,T) est une topologie, alors la topologie (E,T') est plus fine que (E,T), autrement dit $T \subset T'$ et que si (E,T) est une topologie *d'Alexandroff*, alors $T = T'$.

Par suite pour tout $A \subset E$, $o_{(E,R)}(A) = i_{(E,T)}(A)$, autrement dit l'intérieur d'un ensemble est égal à l'origine de l'ensemble pour le préordre associée à la topologie d'Alexandroff. □

Lemme 2.6 (De Tarski vers Kripke)

Soit $I = (E,T,v)$ une interprétation de Tarski.

Soit (E,R) le préordre associée à la topologie d'Alexandroff (E,T) et soit $J = (E,R,v)$ l'interprétation de Kripke associée à I. Pour toute formule A, $v_T^I(A) = v_K^I(A)$.

Preuve : Le lemme est prouvé par récurrence sur l'ensemble des formules. Il est la conséquence immédiate de ce que, d'après 2.5, pour tout $A \subset E$, on a $o(A) = i(A)$, où $i(A)$ est l'intérieur de A, et $o(A)$ l'origine de A pour *le préordre associé à la topologie*.

□

Théorème 2.7 *Une formule est Kripke-valide si et seulement si elle est Tarski-valide.*

Preuve : Montrons qu'une formule Kripke-valide est Tarski-valide.

Supposons au contraire qu'il y a une formule A qui n'est pas Tarski-valide. Il existe une interprétation $I = (E,T,v)$ de Tarski telle que $v_T^I(A) \neq E$. Soit (E,R) le préordre associé à la topologie d'Alexandroff (E,T) et $J = (E,R,v)$ l'interprétation de Kripke. D'après le lemme 2.6, $v_K^I(A) \neq E$. Donc A n'est pas Kripke valide.

Réciproquement toute formule Tarski-valide est Kripke-valide.

Supposons au contraire qu'il y a une formule A qui n'est pas Kripke-valide. Il existe une interprétation $I = (E,R,v)$ de Kripke telle que $v_T^I(A) \neq E$. Soit (E,T) la topologie d'Alexandroff associée au préordre (E,T) et $J = (E,T,v)$ l'interprétation de Tarski. D'après le lemme 2.3, $v_T^I(A) \neq E$. Donc A n'est pas Kripke valide. □

Théorie des langages

Pierre Berlioux, Mnacho Echenim et Michel Lévy

30 novembre 2009

Table des matières

I Langages, relations **9**
1 Chaînes et langages . . . 9
1.1 Vocabulaire . . . 9
1.2 Chaîne . . . 9
1.2.1 Définition . . . 9
1.2.2 Longueur d'une chaîne . . . 9
1.2.3 Ensemble des chaînes sur un vocabulaire . . . 10
1.3 Sous-chaîne . . . 10
1.4 Concaténation . . . 10
1.5 Monoïde . . . 10
1.6 Monoïde libre . . . 10
1.6.1 Définition . . . 10
1.6.2 Homomorphisme de monoïdes . . . 11
1.6.3 Unicité du monoïde libre . . . 11
1.7 Élévation à la puissance . . . 11
1.8 Langage . . . 11
1.9 Opérations sur les langages . . . 12
1.9.1 Opérations ensemblistes . . . 12
1.9.2 Concaténation et concaténation itérée . . . 12
1.9.3 Expressions régulières . . . 13
2 Relations . . . 14
2.1 Définition . . . 14
2.2 Opérations sur les relations . . . 14
2.2.1 Opérations ensemblistes . . . 14
2.2.2 Composition des relations . . . 15
2.2.3 Fermeture transitive d'une relation . . . 15

II Systèmes de réécriture, Grammaires **17**
1 Définitions des langages . . . 17
2 Systèmes de réécriture . . . 17
2.1 Définition . . . 17
2.2 Relations de dérivation dans un système de réécriture . . . 18
2.3 Exemples de systèmes de réécriture . . . 19
2.4 Propriété (composition des dérivations) . . . 19
3 Grammaires . . . 19
3.1 Définition . . . 19
3.2 Langage engendré par une grammaire . . . 19
3.3 Classification des grammaires . . . 20
3.3.1 Grammaires générales (de type 0) . . . 20
3.3.2 Grammaires sous-contexte (de type 1) . . . 20

3.3.3 Grammaires hors-contexte (de type 2) . . . 20
3.3.4 Grammaires linéaires à droite, grammaires linéaires à gauche (de type 3) . . . 20
3.3.5 Définitions et propriétés . . . 20
3.3.6 Problèmes décidables - problèmes indécidables . . . 21
3.3.7 Choix d'un formalisme (classe de définitions) pour définir un langage de programmation . . . 21
3.4 Exemples . . . 22
3.4.1 Définition des nombres réels sans signe d'un langage de programmation avec une grammaire hors-contexte . . . 22
3.4.2 Langage $\{a^n b^n | n \geq 1\}$ . . . 22
3.4.3 Langage des parenthèses des expressions . . . 22
3.4.4 Grammaire engendrant les expressions régulières sur {a, b} . . . 22
3.4.5 Langage $\{a^n b^n c^n | n \geq 1\}$ . . . 23
3.5 Exercices . . . 23

III Grammaires hors-contexte **27**
1 Propriété fondamentale (décomposition des dérivations hors-contexte) . . . 27
2 Dérivations indicées dans une grammaire hors-contexte . . . 28
2.1 Dérivations indicées gauches . . . 28
2.2 Dérivations gauches . . . 29
3 Décomposition des dérivations indicées gauches . . . 29
4 Arbre de dérivation . . . 30
5 Ambiguïté . . . 32

IV Automates finis **35**
1 Introduction . . . 35
2 Automates finis et langages . . . 37
3 Décomposition des traces . . . 38
4 Automates finis sans ε-transition . . . 38
5 Automates finis déterministes . . . 39
6 Construction des automates déterministes . . . 41
7 Machines de Mealy et de Moore . . . 43

V Equivalence des automates finis et des expressions régulières **45**
1 Tout langage régulier est reconnu par un automate fini . . . 45
2 Tout langage reconnu par un automate fini est un langage régulier . . . 48
3 Système d'équations associé à un automate fini . . . 50

VI Automates finis déterministes minimaux **53**
1 Construction de l'automate minimal . . . 53
2 Calcul de la relation d'équivalence entre les états d'un automate . . . 55
3 Unicité de l'automate minimal . . . 57
3.1 Propriétés de la catégorie des automates . . . 57
4 Automate minimal et théorème de Myhill-Nérode . . . 59

VIIDéfinition de langages par induction **65**
1 Exemples de définitions inductives de langages . . . 65
2 Théorèmes du point fixe . . . 66
2.1 Ordre faiblement complet . . . 66
2.2 Ordre complet . . . 67
2.3 Treillis finis . . . 68
3 Langages algébriques . . . 68
4 Équivalence entre langages algébriques et hors-contextes . . . 71

5 Transformation de systèmes d'équations . . . 72
5.1 Substitution simple . . . 73
5.2 Substitution générale . . . 74

VIII Transformations des grammaires hors-contexte **77**
1 Rappels du chapitre précédent . . . 77
1.1 Grammaires et équations . . . 77
1.2 Des substitutions, qui préservent les langages . . . 78
1.3 Treillis fini et calcul de points fixes . . . 78
2 Réduction d'une grammaire hors-contexte . . . 78
2.1 Calcul de l'ensemble des symboles productifs . . . 78
2.2 Calcul de l'ensemble de l'ensemble des symboles accessibles . . . 79
2.3 Construction d'une grammaire réduite . . . 80
3 Élimination des ε-règles . . . 81
4 Élimination des 1-règles . . . 83
5 Forme normale de Chomsky . . . 84
6 Grammaires hors-contexte récursives . . . 85
7 Élimination des règles directement récursive à gauche . . . 86
8 Élimination de la récursivité à gauche . . . 87
9 Forme normale de Greibach . . . 89

IX Conditions d'unicité des systèmes d'équation **93**
1 Système strict . . . 93
2 Conditions nécessaires et suffisantes d'unicité . . . 94
3 Problèmes de décision . . . 96
3.1 Le problème «un système est-il riche ?» est indécidable . . . 97
3.1.1 Système riche : rappel . . . 97
3.2 Indécidabilité du problème «un système est-il riche ?» . . . 97
3.2.1 Le problème «un système a-t-il une seule solution ?» est relativement décidable . . . 98

References **99**

Introduction

Le terme «langage» désigne d'une part les langues naturelles - sous leur forme parlée et écrite, d'autre part les systèmes de notations - les formalismes - utilisés dans diverses sciences comme les mathématiques, la logique, la chimie, etc. et en particulier en informatique (les langages de programmation par exemple). Tout langage comprend un ensemble d'objets élémentaires, que l'on peut composer pour constituer des unités qui ont un sens. La composition des objets élémentaires est le plus souvent un simple enchaînement linéaire (une concaténation) ; les unités qui ont un sens sont alors des suites finies d'objets élémentaires. Cette propriété pour un même langage peut se répéter à plusieurs niveaux. Ainsi dans une langue naturelle écrite, un mot est une suite de lettres (niveau lexical) ; une phrase est une suite de mots (niveau syntaxique) ; un texte est une suite de phrases (niveau du discours).

La définition d'un langage à un niveau donné est la définition

1. de l'ensemble des objets élémentaires (le «vocabulaire» ou l'«alphabet» du langage du niveau considéré)
2. de l'ensemble des suites d'objets élémentaires ayant un sens (la «syntaxe»)
3. des sens de ces suites (la «sémantique»). La définition d'un langage utilise elle-même un langage (souvent appelé «métalangage»), qui peut être une langue naturelle ou un formalisme. Le langage défini est parfois utilisé comme métalangage.

Exemple 1 : le français écrit.
Niveau lexical (ou lexicographique) : les objets élémentaires sont les lettres de l'alphabet, les unités ayant un sens des mots. Un dictionnaire comme le Robert donne une définition lexicographique de français : énumération des mots, définition - en français - des sens de chaque mot. Un dictionnaire français-anglais donne aussi une définition lexicographique du français, les sens des mots français étant définis par leurs traductions en anglais. Au niveau *morphologique*, les mots sont considérés comme des suites d'objets élémentaires qui sont eux-mêmes des suites de lettres appartenant à des catégories grammaticales comme préfixe, suffixe, racine, désinence, etc.
Niveau syntaxique : les objets élémentaires sont les mots et les signes de ponctuation, les unités ayant un sens les phrases. Une grammaire comme le «bon usage» de Grevisse donne une définition du français au niveau syntaxique : elle définit les phrases correctes et leur sens au moyen de règles exprimées en français, avec des termes techniques comme «adjectif», «substantif», «masculin», «subordonnée», «relative», etc. La définition des phrases correctes sans référence à leurs sens constitue l'aspect *formel* (ou syntaxique proprement dit) de la grammaire.

La définition du sens des phrases constitue la *sémantique*. Notons qu'au niveau syntaxique les objets élémentaires (les mots) ont un sens, et que le sens d'une phrase est fonction du sens des mots qui la composent. Il en est de même au niveau morphologique, où le sens d'un mot est fonction des sens des objets élémentaires (préfixes, racines,etc) qui le composent. Par contre au niveau lexical, les lettres n'ont pas de sens, et le sens d'un mot est arbitraire.

Exemple 2 : le français parlé.
Un mot (ou une phrase) prononcé est une suite continue de sons. L'alphabet phonétique définit un ensemble de notations de sons élémentaires tel que l'on puisse considérer tout mot (ou toute phrase) prononcé comme une suite discrète de tels objets élémentaires.

Exemple 3 : un langage de programmation (Ada, C, Java, etc.)
Les objets élémentaires sont les symboles de base (lettres de l'alphabet, chiffres, caractères spéciaux) dont la liste

est donnée explicitement. Les unités ayant un sens sont les programmes. Ceux-ci sont définis par des diagrammes syntaxiques (ou des formes normales de Backus) complétées par des conditions supplémentaires (par exemple sur les déclarations des identificateurs) qui sont souvent exprimées en langue naturelle. Le sens des programme est lui aussi souvent défini en langue naturelle. Une telle définition est un mélange de définitions formelles (les diagrammes syntaxiques) et de définitions informelles (les paragraphes en langue naturelle). On peut aussi donner des définitions formelles (c'est-à-dire n'utilisant pas la langue naturelle) des langages de programmation. Par exemple le sens d'un programme peut-être défini en lui associant une formule mathématique définisssant la fonction qu'il calcule, ou un programme équivalent dans un autre langage de programmation - ainsi un compilateur définit le sens d'un programme par traduction dans un langage machine. On peut encore définir formellement la sémantique d'un langage au moyen d'un interpréteur de ce langage, le sens d'un programme est alors l'ensemble de ses exécutions pour toutes les données possibles. L'interpréteur est un programme qui peut être écrit dans un autre langage de programmation ou dans le langage défini.

Notons que le mot «formel» signifie couramment «exprimé en langage mathématique», ou plus généralement «exprimé dans un formalisme» (c'est-à-dire dans un langage artificiel rigoureusement défini), «informel» signifiant «exprimé en langue naturelle» (c'est ainsi qu'il faut comprendre les termes «définition formelle», «définition informelle») ; cependant on entend par *langage formel* un langage considéré comme un ensemble de suites d'objets élémentaires, abstraction faite du sens de ces suites. Cet aspect formel de la définition d'un langage est aussi appelé aspect syntaxique et doit être complété par la définition du sens des unités du langage, ce qui constitue l'aspect sémantique.

L'objet de ce cours est l'étude de formalismes utilisés pour la définition syntaxiques de langages artificiels comme les langages de programmation ou les systèmes de notations mathématiques et logiques. Ces formalismes sont aussi utilisés pour étudier les langues naturelles, mais ces applications ne seront pas développées.

La première partie de ce polycopié est une *introduction à la théorie des langages formels*. Un langage formel est un ensemble de suites d'objets élémentaires, qui selon le langage défini, et le niveau auquel on se place, peuvent être des lettres, des mots, des symboles, des caractères, etc. Tous ces ensembles d'objet élémentaires ont en commun le fait d'être finis, et sont appelés *vocabulaire* (ou *alphabet*). Un langage formel est un ensemble de suites finies (de *chaînes*) d'éléments de son vocabulaire. Les opérations de base sur les chaînes et les langages sont définies au chapitre I, où sont aussi données des propriétés élémentaires de la notion de relation, qui joue un rôle important dans la suite. La notion de langage comme ensemble de chaînes est trop générale : un langage doit de plus satisfaire des conditions d'effectivité pour pouvoir être traité par des machines physiques (cas des langages de programmation), ou abstraite (cas des systèmes de notations logiques), ou être compris et énoncé par une personne (cas des langues naturelles, des systèmes de notations mathématiques, des langages de programmation évolués, etc). La notion de système de réécriture est la notion de base pour la définition des machines abstraites comme les machines de Turing, et des automates et des grammaires utilisés pour la reconnaissance et la génération des langages formels (chapitre II). Les grammaires les plus utilisées sont les grammaires hors-contexte, qui sont étudiées au chapitre III. On étudie ensuite (chapitres IV,V,VI) les automates finis et les langages réguliers.

La deuxième partie (chapitre VII : *définition d'ensembles par récurrence*) comprend une étude des théorèmes du point fixe qui sont à la base de telles définitions et une étude des systèmes d'équations définissant les langages hors-contexte.

La troisième partie (*compléments sur les langages hors-contexte*), comprend une étude des grammaires hors-contexte et de leurs formes normales (chapitre VIII), une étude de propriétés importantes des langages hors-contextes (chapitre IX), et une étude des automates à pile, qui sont les automates associés aux grammaires hors-contextes (chapitre X).

La quatrième partie (*Introduction à la théorie de la calculabilité*) comprend un chapitre où sont définies les machines de Turing et démontrées leurs propriétés de base (chapitre XI), et un chapitre consacré aux problèmes de décision pour les langages hors-contextes et les langages récursivement énumérables (chapitre XII).

Chapitre I

Langages, relations

1 Chaînes et langages

1.1 Vocabulaire

On appelle vocabulaire (ou alphabet) un ensemble fini quelconque. Les éléments d'un vocabulaire sont appelés lettres, caractères ou symboles. On note #V le nombre d'éléments d'un vocabulaire V.

1.2 Chaîne

1.2.1 Définition

On appelle chaîne (ou mot, ou phrase) sur un vocabulaire V toute suite finie $a_1, a_2, \ldots, a_n$ d'éléments de V. On note usuellement une chaîne en écrivant en séquence, sans séparateur, les lettres qui la composent.

Exemple 1.1 *aba est une chaîne sur le vocabulaire {a, b}.*
<u>début</u>x := 1<u>fin</u> est une chaîne sur le vocabulaire {<u>début</u>, <u>fin</u>, :=, 1, x}, qu'il est préférable d'écrire en utilisant l'espace comme séparateur : <u>début</u> x := 1 <u>fin</u> .

Remarque 1.1 *Un vocabulaire V peut être un ensemble fini de chaînes, comme dans le second exemple ci-dessus. L'écriture des chaînes sur V sans séparateurs entre les éléments de V peut alors produire des ambiguïtés. Par exemple, si V = {a, aa}, il faut un séparateur pour distinguer les deux chaînes a,aa et aa,a.*

1.2.2 Longueur d'une chaîne

On appelle longueur d'une chaîne le nombre d'éléments de la suite la définissant. La longueur d'une chaîne x sera notée $|x|$.

Avec les exemples ci-dessus, on a : | aba | = 3 et | <u>début</u> x := 1 <u>fin</u> | = 5.

Par convention, il existe une chaîne de longueur 0, la *chaîne vide*, notée ε.

Etant donné un vocabulaire V, on note V^n, où n est un entier positif ou nul, l'ensemble des chaînes sur V de longueur n. On a les propriétés suivantes :

- $V^0 = \{\varepsilon\}$: la chaîne vide est une chaîne sur tout vocabulaire.
- $V^1 = V$: les chaînes de longueur 1 sont identifiées aux lettres.

Exercice 1.1 *Montrer que si $V \neq \emptyset$, V^n comprend $(\#V)^n$ éléments.*

1.2.3 Ensemble des chaînes sur un vocabulaire

Etant donné un vocabulaire V, on note V^* l'ensemble des chaînes sur V, et V^+ l'ensemble des chaînes non vides sur V. Autrement dit, $V^* = \bigcup_{i \geq 0} V^i$ et $V^+ = \bigcup_{i > 0} V^i$.

Propriété 1.1
- *Si $V \neq \emptyset$ alors V^* est infini.*
- $\emptyset^* = \{\varepsilon\}$.

1.3 Sous-chaîne

Etant donné une chaîne w sur un vocabulaire V, une chaîne u est une sous-chaîne de w s'il existe $x, y \in V^*$ tels que $w = xuy$. La sous-chaîne u est un préfixe de w si $x = \varepsilon$; un suffixe de w si $y = \varepsilon$.

Exemple 1.2 *La chaîne abba admet les sous-chaînes* ε*, a, b, ab, bb, ba, abb, bba, abba. Il existe deux occurrences de chacune des sous-chaînes a et b. Les préfixes de abba sont* ε*, a, ab, abb et abba ; ses suffixes sont* ε*, a, ba, bba et abba.*

Exercice 1.2

1. *Déterminer les nombres maximal et minimal de sous-chaînes, de préfixes et de suffixes d'une chaîne de longueur donnée.*
2. *Montrer que toute chaîne sur {a, b}, de longueur supérieure ou égale à 4, admet deux occurrences consécutives d'une même sous-chaîne non vide.*

1.4 Concaténation

La concaténation est l'opération qui associe à deux chaînes x et y la chaîne notée $x.y$ ou xy définie par :
si $x = a_1, a_2, \ldots, a_n$ et $y = b_1, b_2, \ldots, b_p$ alors $x.y = c_1, c_2, \ldots, c_{n+p}$,
avec $c_i = a_i$ pour $i = 1, \ldots n$ et $c_{n+i} = b_i$ pour $i = 1, \ldots p$.
Par exemple, la concaténation des chaînes *aba* et *cab* donne la chaîne *abacab*.
On vérifie facilement que la concaténation est une opération associative admettant la chaîne vide comme élément neutre. Soit :
- $\forall x, y, z \in V^* x.(y.z) = (x.y).z$
- $\forall x \in V^* x.\varepsilon = \varepsilon.x = x$.

1.5 Monoïde

On appelle monoïde un ensemble muni d'une opération associative admettant un élément neutre. Un monoïde est représenté par un triplet $< E, ., i >$, où E est l'ensemble considéré, $''.''$ est l'opération associative sur E et $i \in E$ est l'élément neutre de cette opération.
Pour tout vocabulaire V, $< V^*, ., \varepsilon >$ est donc un monoïde.
Comme exemple d'autres monoïdes, citons $< \mathbb{N}, +, 0 >$ et $< \mathbb{N}, ., 1 >$, où $+$ et . dénotent respectivement l'addition et la multiplication des entiers.

1.6 Monoïde libre

1.6.1 Définition

Soit V un vocabulaire quelconque. Le monoïde V^* possède les deux propriétés suivantes :

1. tout élément de V* est une suite d'éléments de V
2. deux suites distinctes d'éléments de V définissent deux éléments distincts de V*.

La propriété 1 distingue le monoïde V^* par exemple du monoïde $(V \cup \{a\})^*$, avec $a \notin V$.

La propriété 2 distingue le monoïde $\{a,b\}^*$ par exemple du monoïde commutatif M obtenu en postulant que l'opération de concaténation est commutative : les deux suites ab et ba d'éléments de V définissent alors le même élément de M.

V^* est le seul monoïde satisfaisant les propriétés 1 et 2 : on dit que V^* est le monoïde libre sur V.

1.6.2 Homomorphisme de monoïdes

Soit $M =< E, ., i >$ et $M' =< E', ., i' >$ deux monoïdes. On appelle homomorphisme de M dans M' toute application h de E dans E' telle que :

- $h(i) = i'$
- $\forall x, y \in E, h(x.y) = h(x).h(y)$.

Un isomorphisme de monoïdes est un homomorphisme bijectif de monoïdes. Si h est un homomorphisme de V^* dans un monoïde quelconque M, alors h est défini par sa valeur pour chaque lettre a de V. En d'autres termes, si h et h' sont deux homomorphismes de V^* dans M tel que $\forall a \in V, h(a) = h'(a)$, h et h' sont identiques. Il en résulte immédiatement la propriété suivante :

Propriété 1.2 *Soit M un monoïde quelconque et f une application d'un vocabulaire V dans M. Alors il existe un homomorphisme unique h de V^* dans M qui prolonge f, c'est-à-dire tel que $\forall a \in V, h(a) = f(a)$.*

1.6.3 Unicité du monoïde libre

Considérons maintenant un monoïde L satisfaisant la propriété P suivante :
soit M un monoïde quelconque et f une application d'un vocabulaire V dans M ; alors il existe une application injective i de V dans L et un homomorphisme unique h de L dans M tel que $\forall a \in V, h(i(a)) = f(a)$.
L'application i définit une représentation des lettres de V par des éléments de L et l'homomorphisme h étend à tous les éléments de L l'application f.
On peut montrer que deux monoïdes satisfaisant la propriété P sont isomorphes.
Le monoïde V^* satisfait la propriété P et tout monoïde L satisfaisant P est alors isomorphe à V^*. On peut donc utiliser la propriété P pour donner une définition algébrique de la notion de monoïde libre sur un vocabulaire V. Cette notion est alors définie à un isomorphisme près (c'est-à-dire à la représentation près des chaînes sur V ; ceci était implicite dans la première définition).

Exercice 1.3 *Montrer que l'application «longueur d'une chaîne» est un homomorphisme de monoïde de $< V^*, ., \varepsilon >$ dans $< \mathbb{N}, +, 0 >$.*
Cette application est un isomorphisme si $\#V = 1$: si V ne contient qu'un élément, $< \mathbb{N}, +, 0 >$ est le monoïde libre sur V car on peut alors représenter sans ambiguïté les chaînes sur V par leurs longueurs.

1.7 Élévation à la puissance

On pose, pour toute chaîne x sur un vocabulaire V,

- $x^0 = \varepsilon$
- pour tout entier $n > 0$, $x^n = x.x^{n-1} = x.x \ldots x$ avec n facteurs.

Comme pour tout monoïde multiplicatif, on a pour tout n, m les propriétés :

- $x^{n+m} = x^n.x^m$
- $x^{nm} = (x^n)^m$

1.8 Langage

On appelle *langage* sur un vocabulaire V tout sous-ensemble de V^*.

Exemple 1.3

- *Un langage de programmation est un langage sur l'ensemble de ses symboles de base.*

- *L'ensemble des phrases de la langue française est un langage sur l'ensemble des mots du français ; l'ensemble des mots du français est lui-même un langage sur l'ensemble des lettres de l'alphabet.*
- *L'ensemble des identificateurs est un langage sur* $\{A,B,...,Z,a,b,...,z,0,1,...,9\}$.
- $\emptyset, V^1, V^2, \ldots, V^n, V^*$ *sont des langages sur* V. *On appelle* $\emptyset$ *le langage vide.*

Un langage est dit fini ou infini selon qu'il comprend un nombre fini ou infini de chaînes.

Propriété 1.3 *Un langage L est infini si et seulement si pour tout* $n \in \mathbb{N}$, *il existe* $x \in L$ *tel que* $|x| > n$.
Soit encore, de façon équivalente : un langage L est fini si et seulement si il existe une borne à la longueur des chaînes de L.

Cette propriété résulte immédiatement de ce que tout vocabulaire est un ensemble fini.

Exemple 1.4 *Le langage des identificateurs de Java est infini car il n'y a pas, dans la définition de Java, de borne pour la longueur des identificateurs.*

1.9 Opérations sur les langages

1.9.1 Opérations ensemblistes

Les langages sur V étant les parties de V^*, on définit de la manière usuelle la réunion $L_1 \cup L_2$, l'intersection $L_1 \cap L_2$, la différence $L_1 - L_2$ et le complémentaire $\overline{L} = V^* - L$ de langages L_1, L_2 et L sur V.

1.9.2 Concaténation et concaténation itérée

On étend aux langages la concaténation définie pour les chaînes en posant pour tous langages L_1, L_2 sur V :

$$L1.L2 = \{x.y \mid x \in L_1, y \in L_2\}$$

Exemple 1.5 *Si* $L_1 = \{a^n b^n | n \geq 0\}$ *et* $L_2 = \{c^n d^n | n \geq 0\}$ *alors* $L_1.L_2 = \{a^p b^p c^q d^q | p,q \geq 0\}$.

La concaténation des langages est associative et admet le langage $\{\varepsilon\}$ comme élément neutre.
Autrement dit $< \mathcal{P}(V^*), ., \{\varepsilon\} >$ est un monoïde.

Exercice 1.4

1. *Prouver que, pour tout* $L \in V^*$, $\emptyset.L = L.\emptyset = \emptyset$.
2. *Etudier le monoïde* $< \mathcal{P}(V^*), ., \{\varepsilon\} >$ *quand* $V = \emptyset$.

Remarque 1.2 *Il ne faut pas confondre le langage vide* $\emptyset$, *qui ne contient aucune chaîne, avec le langage* $\{\varepsilon\}$, *qui contient la chaîne vide* ε. *Le premier est l'élément absorbant et le second est l'élément neutre de la concaténation des langages.*

L'élévation à une puissance entière d'un langage L est définie comme pour tout monoïde multiplicatif :
- $L^0 = \{\varepsilon\}$
- pour tout $n > 0, L^n = L.L^{n-1} = L.L \ldots L$ avec n facteurs.

On définit enfin les opérations de concaténation itérée * et + par : $L^* = \bigcup_{i \geq 0} L^i$ et $L^+ = \bigcup_{i > 0} L^i$.
On identifie souvent, quand il n'y a pas d'ambiguïté possible, les langages sur V ne comprenant qu'une seule chaîne avec les chaînes de V^*.
On écrit alors par exemple a^* pour $\{a\}^*$ et ε pour $\{\varepsilon\}$.

Quand le langage L est fini, il peut être considéré comme un vocabulaire. Les notations L^n, L^* et L^+ sont alors ambiguës : elles désignent aussi, respectivement, les chaînes sur L de longueur n, l'ensemble des chaînes sur L et l'ensemble des chaînes non vides sur L (voir les paragraphes 1.2.2 et 1.2.3). Par exemple, si $L = \{a, aa\}$, L^2 est d'une part le langage qui est la concaténation de L à lui-même, d'autre part l'ensemble des chaînes de longueur 2 sur L. Dans le premier cas, on a $L^2 = \{aa, aaa, aaaa\}$, dans le second cas $L^2 = \{a.a, a.aa, aa.a, aa.aa\}$.

1.9.3 Expressions régulières

On appelle *expression régulière* sur un vocabulaire V une expression dont les opérandes sont des lettres de V ou le symbole $\emptyset$ et les opérateurs sont pris dans l'ensemble $\{+,.,*\}$, où $+$ et . sont binaires et $*$ est unaire. On écrit usuellement une expression régulière avec les conventions suivantes :

- on utilise la notation infixée pour les opérateurs binaires $+$ et ., la notation en exposant pour l'opérateur unaire $*$
- les priorités des opérateurs sont dans l'ordre : $*,.,+$.

Une expression régulière définit un langage sur V quand on identifie chaque lettre a de V au langage $\{a\}$, $\emptyset$ au langage vide et que l'on interprète $+$ comme la réunion, . comme la concaténation des langages et $*$ comme la concaténation itérée définie en 1.9.2. On dit encore que le langage ainsi défini est la valeur de l'expression régulière.
On appelle *langage régulier* tout langage qui peut être défini par une expression régulière. En particulier tout langage fini est régulier.

Exemple 1.6

1. $(A+B+\ldots+Z+a+b+\ldots+z).(A+B+\ldots+Z+a+b+\ldots+z+0+1+\ldots+9)^*$ *est une expression régulière définissant le langage des identificateurs de Java.*
2. $\emptyset^*$ *est une expression régulière définissant le langage* $\{\varepsilon\}$.

Exercice 1.5

1. *Montrer que l'ensemble des langages réguliers sur un vocabulaire V est le plus petit ensemble de langages contenant les langages finis sur V et fermé pour la réunion, la concaténation et l'opération étoile.*
2. *Soit L un langage sur un vocabulaire V. Montrer que L^* est le plus petit langage sur V contenant L et ε, fermé pour la concaténation.*
3. *Montrer que les égalités suivantes sont vraies pour tout langage L :*
   - $(L^*)^* = L^*$
   - $(\varepsilon+L)^* = L^*$
   - $L^*.L^* = L^*$
4. *Montrer que, pour tout langage L,M,N et P, si $L \subseteq M$ et $N \subseteq P$, alors $L^* \subseteq M^*$ et $L.N \subseteq M.P$ (en d'autres termes, l'étoile et la concaténation sont des opérations croissantes.)*
5. *Montrer que les égalités suivantes sont vraies pour tous langages L et M :*
   - $(L^*.M^*)^* = (L+M)^*$
   - $(L.M)^*L = L.(M.L)^*$
   - $(L.M+L)^*.L = L.(M.L+L)^*$

   Montrer que l'égalité suivante n'est pas valide : $(L+M)^* = L^*+M^*$
6. *Soit L et M deux langages sur un vocabulaire V, X une inconnue à valeur dans l'ensemble des langages sur V.*
   (a) *Montrer que l'équation $L.X = M$ n'admet pas nécessairement une solution.*
   (b) *Montrer que l'équation $X = L.X+M$ admet toujours une solution.*
   (c) *Donner des exemples de langages L et M tels que l'équation $X = L.X+M$ admet plusieurs solutions.*
   (d) *Reprendre les questions 6b et 6c avec l'équation $X = X.L+M$.*
7. *Combinatoire de mots*
   (a) *Soit V un vocabulaire non vide, x et y deux mots sur V tels que $xy = yx$. Montrer qu'il existe $u \in V^+$ et deux entiers $p \geq 0$ et $q \geq 0$ tels que $x = u^p$ et $y = u^q$.*
   (b) *Soit $x,z \in V^+$, $y \in V^*$ tels que $xy = yz$. Montrer qu'il existe deux mots sur V, u, v et un entier $p \geq 0$ tels que $x = uv$, $z = vu$ et $y = (uv)^p u = u(vu)^p$.*
8. *Soit $V = \{0,1\}$. Donner des expressions régulières sur V définissant les langages suivants :*
   (a) *L'ensemble des chaînes sur V contenant au moins une occurrence de 0.*
   (b) *L'ensemble des chaînes sur V de longueur impaire.*

(c) *L'ensemble des chaînes sur V ne contenant pas deux occurrences consécutives de* 0*, ni deux occurrences consécutives de* 1.

9. *Soit l'opération $\sim$ définie sur V* par :*
   - $\varepsilon^{\sim} = \varepsilon$
   - $\forall a \in V, w \in V^*, (a.w)^{\sim} = w^{\sim}.a$

   (a) *Prouver que* $\forall x, y \in V^*, (xy)^{\sim} = y^{\sim}.x^{\sim}$

   (b) *Prouver que* $\forall w \in V^*, (w^{\sim})^{\sim} = w$

   (c) *Soit $\otimes$ l'opération sur V^* définie par :* $\forall x, y \in V^*, x \otimes y = y.x$
   - *Montrer que* $< V^*, \otimes, \varepsilon >$ *est un monoïde.*
   - *Montrer que l'opération $\sim$ est l'homomorphisme de* $< V^*, ., \varepsilon >$ *dans* $< V^*, \otimes, \varepsilon >$ *qui étend l'application identique de V dans V.*
   - *Montrer que* $< V^*, \otimes, \varepsilon >$ *et* $< V^*, ., \varepsilon >$ *sont isomorphes. Ainsi* $< V^*, \otimes, \varepsilon >$ *est, au même titre que* $< V^*, ., \varepsilon >$*, le monoïde libre sur V : on peut écrire indifféremment les chaînes de gauche à droite ou de droite à gauche.*

   (d) *Etendre l'opération $\sim$ aux langages sur V.*

   (e) *Prouver que, pour tout langage L sur V, si L est régulier, alors $L^{\sim}$ est régulier.*

   (f) *On appelle palindrome sur V toute chaîne w sur V telle que $w^{\sim} = w$. Soit P le langage des palindromes sur V. Montrer que :* $P = \{w.w^{\sim} \mid w \in V*\} \cup \{waw^{\sim} | w \in V^*, a \in V\}$.

2 Relations

2.1 Définition

On appelle relation (binaire) sur un ensemble E toute partie de $E \times E$. Etant donné une relation ρ sur E et deux éléments x et y de E, on dit que ρ est vraie pour le couple (x, y) si et seulement si $(x, y) \in \rho$. Dans la suite, on écrira souvent $x\rho y$ pour $(x, y) \in \rho$.

Exemple 2.1 *Pour tout ensemble E, $\emptyset$ est la relation partout fausse, $E \times E$ est la relation partout vraie sur E. On note ι_E, ou ι s'il n'y a pas d'ambiguïté, la relation identité sur E, c'est-à-dire la partie de $E \times E$ définie par :*
$\iota = \{(x, x) | x \in E\}$ *. La relation ι est aussi notée par le symbole =.*

On dit qu'une relation ρ sur E est *réflexive* si $\forall x \in E, x\rho x$; qu'elle est *transitive* si $\forall x, y, z \in E$, $x\rho y$ et $y\rho z$ impliquent $x\rho z$.
Une relation réflexive et transitive est dite *relation de préordre*.
Une relation ρ sur E est *symétrique* si $\forall x, y \in E$, $x\rho y$ implique $y\rho x$; *antisymétrique* si $\forall x, y \in E$, $x\rho y$ et $y\rho x$ impliquent $x = y$.
Une relation de préordre symétrique est dite *relation d'équivalence* ; une relation de préordre antisymétrique est dite *relation d'ordre partiel* ; une *relation d'ordre (total)* est une relation d'ordre partiel telle que $\forall x, y \in E$, $x\rho y$ ou $y\rho x$.
Une *relation fonctionnelle* sur E (ou fonction partielle de E dans E) est une relation ρ sur E telle que $\forall x, y \in E$, $x\rho y$ et $x\rho z$ impliquent $y = z$; une *fonction totale* (une *application*) de E dans E est une relation fonctionnelle ρ sur E telle que pour tout x de E il existe un y de E tel que $x\rho y$.

2.2 Opérations sur les relations

2.2.1 Opérations ensemblistes

Les relations étant les parties de $E \times E$, on définit de la manière usuelle le complémentaire $\overline{\rho}$, la réunion $\rho \cup \sigma$ et l'intersection $\rho \cap \sigma$ de relations ρ et σ sur E. On a :
- $(x, y) \in \rho \cup \sigma$ si et seulement si $x\rho y$ ou $x\sigma y$
- $(x, y) \in \rho \cap \sigma$ si et seulement si $x\rho y$ et $x\sigma y$

– $x\overline{\rho}y$ si et seulement si $(x,y) \notin \rho$ c'est-à-dire non $x\rho y$

Etant donné deux relations ρ et σ sur E, on dit que ρ implique σ (que σ contient ρ) si et seulement si $\rho \subseteq \sigma$, c'est-à-dire si et seulement si pour tout x et y de E, $x\rho y$ implique $x\sigma y$.

On vérifie facilement que ρ implique σ si et seulement si $\overline{\rho} \cup \sigma = E \times E$.

2.2.2 Composition des relations

La composition des relations sur E est l'opération sur $\mathcal{P}(E \times E)$ définie par :
$\forall \rho, \sigma \in \mathcal{P}(E \times E), \rho.\sigma = \{(x,y) \in E \times E \mid \exists z \in E, (x,z) \in \rho \text{ et } (z,y) \in \sigma\}$.

La composition des relations est associative et admet comme élément neutre la relation identité ι. Autrement dit, $< \mathcal{P}(E \times E), ., \iota >$ est un monoïde.

Exercice 2.1 *Montrer qu'une relation ρ sur E est transitive si et seulement si $\rho.\rho \subseteq \rho$.*

On définit pour toute relation ρ sur E et tout entier $n \geq 0$ la relation ρ^n , comme pour tout monoïde multiplicatif, par :

– $\rho^0 = \iota$
– $\forall n > 0, \rho^n = \rho.\rho^{n-1}$

On vérifie immédiatement que, pour tout $n > 0$, $x\rho^n y$ si et seulement si il existe une suite $x_1, \ldots, x_{n+1}$ d'éléments de E tels que $x_1 = x, x_{n+1} = y$ et pour tout i, $1 \leq i \leq n, x_i \rho x_{i+1}$.

On définit enfin pour toute relation ρ les relations ρ^* et ρ^+ par :

$$\rho^* = \bigcup_{n \geq 0} \rho^n \text{ et } \rho^+ = \bigcup_{n > 0} \rho^n$$

Il est clair que :

– $x\rho^* y$ si et seulement si $x = y$ ou $x\rho^+ y$
– $x\rho^+ y$ si et seulement si il existe un entier $n > 0$ et une suite $x_1, \ldots, x_{n+1}$ d'éléments de E tels que $x_1 = x, x_{n+1} = y$ et pour tout i, $1 \leq i \leq n, x_i \rho x_{i+1}$

2.2.3 Fermeture transitive d'une relation

On appelle fermeture transitive d'une relation ρ sur E la plus petite relation transitive sur E contenant ρ, c'est-à-dire la relation sur E qui contient ρ, qui est transitive et qui est incluse dans toute relation transitive contenant ρ.

Proposition 2.1 *Pour toute relation ρ sur un ensemble E, ρ^+ est la fermeture transitive de ρ .*

Preuve : Il est clair que ρ^+ est transitive. Soit maintenant σ une relation transitive contenant ρ. Si $x\rho^+ y$, alors il existe $x_2, \ldots, x_n$, avec $n > 0$, tels que $x\rho x_2 \ldots x_n \rho y$. Comme σ contient ρ, on a $x\sigma x_2 \ldots x_n \sigma y$ et, σ étant transitive,$x\sigma y$.
□

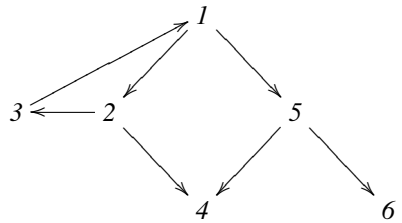
On appelle *fermeture transitive et réflexive* d'une relation ρ la plus petite relation transitive et réflexive contenant ρ.

Proposition 2.2 *Pour toute relation ρ sur un ensemble E, ρ^* est la fermeture transitive et réflexive de ρ.*

Preuve : exercice. □

Exercice 2.2

1. *Soit ρ la relation sur l'ensemble {1, 2, 3, 4, 5, 6} définie par le graphe de la figure ci-dessous. Tracer les graphes des relation ρ^+ et ρ^* .*



2. *Définir un algorithme qui, étant donné une relation sur un ensemble fini, calcule sa fermeture transitive.*

Chapitre II

Systèmes de réécriture, Grammaires

1 Définitions des langages

On définit couramment un langage, c'est-à-dire un ensemble de chaînes sur un vocabulaire V, par une expression du type $\{x \in V^* \mid P(x)\}$, où P est une propriété des chaînes sur V.

Par exemple $\{x \mid \exists n \in \mathbb{N}, x = a^n b^n\}$, c'est-à-dire $\{a^n b^n \mid n \geq 0\}$; $\{x \mid x = ab$ ou $x = cd\}$, c'est-à-dire $\{ab, cd\}$; $\{x \mid x$ appartient au langage défini par l'expression régulière $a(a+b)^*\}$, que l'on note simplement $a(a+b)^*$.

Certaines définitions permettent de construire un algorithme de reconnaissance du langage défini, c'est-à-dire un algorithme distinguant les chaînes du langage défini des autres chaînes sur le vocabulaire. Plus précisément, ces définitions appartiennent à une classe de définitions qui peuvent être traduites (compilées) en algorithmes de reconnaissance du langage défini. C'est le cas de toute définition d'un langage fini par la donnée explicite de ses chaînes : on peut alors construire une table des éléments du langage. C'est encore le cas de toute définition d'un langage par une expression régulière : il est en effet possible de construire à partir de toute expression régulière un algorithme de reconnaissance du langage défini (voir le chapitre V). L'intérêt pratique de définitions de ce type est évident : le fait que l'on puisse déduire de manière systématique de la définition d'un langage de programmation un algorithme de reconnaissance des programmes facilite l'écriture de compilateurs pour ce langage. On étudiera dans la suite différentes classes de définitions de langages, que l'on peut traduire en algorithmes de reconnaissance des langages définis. Ces modes de définitions utilisent souvent la notion de système de réécriture, qui est introduite au paragraphe 2 de ce chapitre. Les grammaires, introduites au paragraphe 3, utilisent un système de réécriture pour engendrer toutes les chaînes du langage défini à partir d'une chaîne donnée, l'axiome de la grammaire. Une autre classe de définitions de langages fondées sur des systèmes de réécriture est constituée par les automates et les machines, dont on verra des exemples au chapitre IV. On étudiera aussi des modes de définitions de langages qui n'utilisent pas de système de réécriture : on peut ainsi définir un langage comme la valeur d'une expression (par exemple d'une expression régulière), ou comme solution d'un système d'équations (voir le paragraphe du chapitre V et le chapitre IX).

2 Systèmes de réécriture

2.1 Définition

On appelle *système de réécriture* un couple $S = < V, R >$ où :

- V est un vocabulaire
- R est un ensemble fini de couples de chaînes sur V

Tout élément (α, β) de R est appelé *règle de réécriture* et est noté $\alpha \rightarrow \beta$; α est la *partie gauche*, β la *partie droite* de la règle $\alpha \rightarrow \beta$. Des règles $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \ldots, \alpha \rightarrow \beta_k$ ayant même partie gauche sont souvent notées
$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_k$.

2.2 Relations de dérivation dans un système de réécriture

Etant donné un système de réécriture $S =< V, R >$ et deux chaînes x et y de V^*, *x se réécrit directement y* si et seulement si il existe une règle $\alpha \rightarrow \beta$ dans R telle que $x = u\alpha v, y = u\beta v$.

On dit encore que x se réécrit y par application de la règle $\alpha \rightarrow \beta$. La partie gauche α de la règle appliquée est une sous-chaîne de x et y est obtenue à partir de x en remplaçant une occurrence de α dans x par la partie droite β.

On note x se réécrit directement y par $x \Rightarrow_S y$, ou $x \Rightarrow_R y$, ou encore, s'il n'y a pas d'ambiguïté sur le système de réécriture utilisé, $x \Rightarrow y$.

Si $x \Rightarrow_R y$, il est possible que plusieurs règles de R permettent de réécrire x en y.
Par exemple, soit $R = \{ab \rightarrow ba, aab \rightarrow aba\}$: on a $aab \Rightarrow_R aba$, par application de la première ou de la deuxième règle de R. Il est aussi possible qu'une même règle puisse s'appliquer à différentes sous-chaînes de x pour donner y ; soit par exemple la règle $a \rightarrow aa$: la chaîne aaa peut être obtenue en réécrivant la première ou la deuxième lettre de la chaîne aa.

La relation $\Rightarrow$ associée à un système de réécriture $< V, R >$ est une relation sur V^*, c'est-à-dire une partie de $V^* \times V^*$. On définit comme pour toute relation (voir paragraphe 2 du chapitre 1) les relations $\Rightarrow^n$, pour n entier positif ou nul, $\Rightarrow^*$ et $\Rightarrow^+$. Pour tout x, y de V^*, on a :

- $x \Rightarrow^n y$ signifie : x se réécrit en y par application de n règles (en particulier, $x \Rightarrow^0 y$ si et seulement si $x = y$)
- $x \Rightarrow^* y$ signifie : x se réécrit y (par application d'un nombre quelconque, éventuellement nul, de règles)
- $x \Rightarrow^+ y$ signifie : x se réécrit y par application d'au moins une règle.

On appelle dérivation dans un système de réécriture toute suite finie $x_1, x_2, \ldots, x_k$ de chaînes telles que $k \geq 1$ et $x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow x_k$. On dit que x_k est dérivée à partir de x_1 : x_1 est l'origine, x_k est l'extrémité de la dérivation. *La longueur* de la dérivation est égale à $k-1$, c'est-à-dire au nombre d'applications de règles de réécriture dans la dérivation. Les chaînes x_i sont les éléments de la dérivation. On a évidemment :

- $x \Rightarrow^* y$ si et seulement si il existe une dérivation d'origine x et d'extrémité y.
- $x \Rightarrow^n y$ si et seulement si il existe une dérivation de longueur n de x à y.

La donnée d'une dérivation ne permet pas toujours de retrouver les règles de réécriture appliquées, ni les sous-chaînes réécrites (car la relation $\Rightarrow$ ne le permet pas toujours). On introduit maintenant la notion de dérivation indicée, qui décrit complètement le processus de réécriture. Etant donné un entier $i \geq 1$ et une règle r, on note $\Rightarrow_{i,r}$ la relation sur V^* définie par :
$x \Rightarrow_{i,r} y$ si et seulement si

- $x = a_1 \ldots a_n$, avec $a_1, \ldots, a_n \in V$,
- $r = a_i \ldots a_k \rightarrow \omega$, $1 \leq i \leq n+1$, $i-1 \leq k \leq n$
- $y = a_1 \ldots a_{i-1} \omega a_{k+1} \ldots a_n$.

L'entier i désigne le premier caractère de la sous-chaîne réécrite de x, et r la règle appliquée.
On a $k = i-1$ quand la partie gauche de la règle r est la chaîne vide et $i = n+1$ quand la chaîne vide est réécrite ω à droite de la chaîne x.
Une dérivation indicée est une suite finie $x_1, < i_1, r_1 >, x_2, < i_2, r_2 >, \ldots, x_{k-1}, < i_{k-1}, r_{k-1} >, x_k$, telle que :

- $k \geq 1$, $x_1, \ldots, x_k \in V^*$,
- $r_1, \ldots, r_{k-1}$ appartiennent à R
- $x_1 \Rightarrow_{i_1, r_1} x_2 \ldots x_{k-1} \Rightarrow_{i_{k-1}, r_{k-1}} x_k$

Comme ci-dessus, la dérivation est *d'origine* x_1, *d'extremité* x_k et de longueur $k-1$.
On associe ainsi à un système de réécriture quatre relations de dérivation :

- $\Delta_1 : x \Rightarrow^* y$ (il existe une dérivation de x à y)
- $\Delta_2 : x \Rightarrow^n y$ (il existe une dérivation de x à y de longueur n)
- $\Delta_3 : x, x_2, \ldots x_n, y$ est une dérivation de x à y
- $\Delta_4 : x, < i_1, r_1 >, x_2, < i_2, r_2 >, \ldots, x_n, < i_n, r_n >, y$ est une dérivation indicée de x à y.

On a les propriétés suivantes :

- Δ_1 s'écrit $\exists n$ tel que Δ_2
- Δ_2 s'écrit $\exists x_2, \ldots x_n$ tels que Δ_3
- Δ_3 s'écrit $\exists i_1, r_1, \ldots i_n, r_n$ tels que Δ_4

2.3 Exemples de systèmes de réécriture

1. $V_1 = \{a,b,0,1,r,s\}, R_1 = \{ra \to s, rb \to s,\ sa \to s,\ sb \to s,\ s0 \to s,\ s1 \to s\}$.
   On vérifie facilement que pour tout w dans $\{a,b,0,1\}^*$, on a $rw \Rightarrow^* s$ si et seulement si le premier caractère de w est a ou b. Ce système de réécriture reconnaît ainsi les identificateurs sur le vocabulaire $\{a,b,0,1\}$.
2. $V_2 = \{0,1,+,E\}, R_2 = \{E \to 0 \mid 1 \mid E+E\}$.
   Pour tout w appartenant à $\{0,1,+\}^*$, on a $E \Rightarrow^* w$ si et seulement si w est une expression construite avec les opérandes 0 et 1 et l'opérateur +.
3. $V_3 = \{0,1,+\}, R_3 = \{0+0 \to 0,\ 0+1 \to 1,\ 1+0 \to 1,\ 1+1 \to 0\}$.
   Ce système de réécriture calcule la somme modulo 2 d'une suite d'entiers binaires en ce sens que si w appartenant à $\{0,1,+\}^*$ est expression dérivée à partir de E dans le système de l'exemple précédent, on a dans R_3 la dérivation $w \Rightarrow^* 0$ ou $w \Rightarrow^* 1$ suivant la valeur de l'expression quand l'opérateur $+$ est interprété comme l'addition modulo 2.
4. $V_4 = \{0,1\}, R_4 = \{10 \to 01\}$.
   Soit $x \in \{0,1\}^*$ et soit y telle que a) $x \Rightarrow^* y$ et b) la règle de R_4 ne s'applique pas à y (en d'autres termes : y ne contient pas d'occurrence de 10). Alors y est la permutation des lettres de x où tous les 0 sont avant les 1. Ce système de réécriture trie donc les chaînes de $\{0,1\}^*$ par chiffres croissants.

2.4 Propriété (composition des dérivations)

Soit $S =< V,R >$ un système de réécriture. Soit $u_1,\ldots,u_n,v_1,\ldots,v_n$ des chaînes de V^*.
Alors si $u_1 \Rightarrow^* v_1,\ldots,u_n \Rightarrow^* v_n$, on a $u_1\ldots u_n \Rightarrow^* v_1\ldots v_n$.
Cette propriété se démontre comme suit :

1. si $u_1 \Rightarrow^* v_1$, on a alors $u_1u_2 \Rightarrow^* v_1u_2$ (démonstration facile par récurrence sur la longueur de la dérivation).
2. On a de même : si $u_2 \Rightarrow^* v_2$, alors $v_1u_2 \Rightarrow^* v_1v_2$.
3. On en conclut que si $u_1 \Rightarrow^* v_1$ et $u_2 \Rightarrow^* v_2$, alors $u_1u_2 \Rightarrow^* v_1v_2$.

La généralisation au cas de n dérivations est immédiate.
Nous énonçons la propriété avec la relation de dérivation Δ_2 (c'est-à-dire $\Rightarrow^n$) :
pour tout $u_1,\ldots,u_n,v_1,\ldots,v_n$ appartenant à V^*, pour tout $q_1,\ldots,q_n$ appartenant à $\mathbb{N}$,
si $u_1 \Rightarrow^{q_1} v_1,\ldots,u_n \Rightarrow^{q_n} v_n$ alors $u_1\ldots u_n \Rightarrow^p v_1\ldots v_n$, avec $p = \Sigma_{i=1}^n q_i$

Exercice 2.1 *Enoncer la propriété de composition pour les relations de dérivation Δ_3 et Δ_4 (c'est-à-dire pour les dérivations et les dérivations indicées). La dérivation d'origine $u_1\ldots u_n$ et d'extrémité $v_1\ldots v_n$ est-elle unique ?*

3 Grammaires

3.1 Définition

On appelle *grammaire* tout quadruplet $G =< V_T, V_N, S, R >$ où

- V_T et V_N sont deux vocabulaires disjoints.
  V_T est le *vocabulaire terminal* (ses éléments sont appelés *symboles terminaux*), V_N est le *vocabulaire non-terminal* (ses éléments sont les *symboles non-terminaux*). Les chaînes sur V_T sont dites *chaînes terminales*.
  On pose $V = V_T \cup V_N$ et V est appelé le vocabulaire de la grammaire.
- S est un élément de V_N, l'*axiome* de la grammaire.
- $< V,R >$ est un système de réécriture. R est l'ensemble des règles de la grammaire.

3.2 Langage engendré par une grammaire

On appelle langage engendré par une grammaire $G =< V_T, V_N, S, R >$ l'ensemble des chaînes sur le vocabulaire terminal V_T que l'on peut dériver à partir de l'axiome S. On note $L(G)$ le langage engendré par la grammaire G. On a

donc : $L(G) = \{x \in V_T^* \mid S \Rightarrow_R^* x\}$.
Des exemples de grammaires sont donnés au paragraphe 3.4.

Définition 3.1 *Deux grammaires sont dites équivalentes si elles engendrent le même langage.*

3.3 Classification des grammaires

En imposant des restrictions sur la forme des règles, on définit différents types de grammaires.

3.3.1 Grammaires générales (de type 0)

Une grammaire est dite *générale* si toutes ses règles sont de la forme :
$\alpha \rightarrow \beta$, avec $\alpha, \beta \in V^*$, et $\alpha \neq \varepsilon$.

3.3.2 Grammaires sous-contexte (de type 1)

Une grammaire est dite *sous-contexte* si toutes ses règles sont de la forme :
$\alpha A \beta \rightarrow \alpha\omega\beta$, avec $\alpha, \beta \in V^*$, $A \in V_N$, $\omega \in V^+$.

3.3.3 Grammaires hors-contexte (de type 2)

Une grammaire est dite *hors-contexte* si toutes ses règles sont de la forme :
$A \rightarrow \omega$, avec $A \in V_N$, $\omega \in V^*$.
On appelle ε-*règle* une règle de la forme $A \rightarrow \varepsilon$, et *1-règle* une règle de la forme $A \rightarrow B$ avec $B \in V_N$.
Les grammaires hors-contexte sont souvent appelées *formes de Bacckus*. Elles sont équivalentes aux cartes syntaxiques couramment utilisées pour définir la syntaxe des langages de programmation.

3.3.4 Grammaires linéaires à droite, grammaires linéaires à gauche (de type 3)

Une grammaire est dite *linéaire à droite* si toutes ses règles sont de l'une des deux formes suivantes :
- $A \rightarrow \omega B$, avec $A, B \in V_N$, $\omega \in V_T*$
- $A \rightarrow \omega$, avec $A \in V_N$, $\omega \in V_T*$

Une grammaire est dite *linéaire à gauche* si toutes ses règles sont de l'une des deux formes suivantes :
- $A \rightarrow B\omega$, avec $A, B \in V_N$, $\omega \in V_T*$
- $A \rightarrow \omega$, avec $A \in V_N$, $\omega \in V_T*$

3.3.5 Définitions et propriétés

- Toute grammaire linéaire à droite, ainsi que toute grammaire linéaire à gauche est une grammaire hors-contexte. Toute grammaire hors-contexte ne contenant pas d'ε-règle est une grammaire sous-contexte.
- Un langage est dit *linéaire à droite (hors-contexte, sous-contexte)* s'il existe une grammaire linéaire à droite (hors-contexte, sous-contexte) qui l'engendre.
- Pour tout langage régulier (c'est-à-dire défini par une expression régulière - voir le paragraphe 1.9.3 du chapitre I), il existe une grammaire linéaire à droite et une grammaire linéaire à gauche qui l'engendrent. Les langages réguliers sont aussi définis les automates finis (voir le chapitre IV). Tout langage régulier est hors-contexte (puisque toute grammaire linéaire à droite est hors-contexte) ; la réciproque est fausse : par exemple le langage $\{a^n b^n \mid n \geq 1\}$ est hors-contexte mais n'est pas régulier.
- Tout langage hors-contexte ne contenant pas la chaîne vide est sous-contexte (en effet, tout langage hors-contexte ne contenant pas ε peut être engendré par une grammaire hors-contexte sans ε-règle ; cette propriété est démontrée au chapitre VIII). La réciproque est fausse : par exemple le langage $\{a^n b^n c^n \mid n \geq 1\}$ est sous-contexte mais n'est pas hors-contexte (voir le chapitre VIII).

– Les langages engendrés par les grammaires générales (c'est-à-dire sans restriction sur les règles) sont dits récursivement énumérables : à partir d'une grammaire, on peut construire un algorithme qui énumère toute les chaînes du langage engendré par la grammaire ; réciproquement, étant donné un algorithme énumérant les chaînes d'un langage, on peut construire une grammaire engendrant ce langage .
– On appelle langage récursif tout langage pour lequel il existe un algorithme de reconnaissance (c'est-à-dire un algorithme qui, pour toute chaîne w sur le vocabulaire du langage, décide si w appartient au langage ou non). On peut montrer que tout langage récursif est récursivement énumérable, et que la réciproque est fausse. Il existe d'autre part des langages qui ne sont pas récursivement énumérables. Les langages sous-contexte (et a fortiori hors-contexte et réguliers) sont récursifs.

Exercice 3.1

1. *Définir un algorithme qui étant donné une grammaire quelconque, énumére les chaînes du langage engendré par cette grammaire.*
2. *Déduire de l'algorithme ci-dessus que pour tout langage récursivement énumérable L il existe un algorithme de reconnaissance partielle (c'est-à-dire un algorithme qui reconnait toute chaîne w de L, mais ne s'arrête pas si w n'appartient pas à L).*
3. *Prouver que tout langage engendré par une grammaire dont les règles satisfont la contrainte « $\alpha \rightarrow \beta$ implique $\mid \alpha \mid \leq \mid \beta \mid$ » est récursif.*
   On peut montrer que ces grammaires sont équivalentes aux grammaires sous-contexte.

3.3.6 Problèmes décidables - problèmes indécidables

Etant donné une grammaire G (linéaire à droite, hors-contexte, etc.), on s'intéressera par la suite à des problèmes du genre :
– $L(G)$ est-il vide ? (problème du langage vide)
– $L(G)$ est-il infini ? (problème du langage infini)
– $L(G) = V_T^*$?
– $L(G)$ est-il un langage régulier ? , etc.
On dit qu'un de ces problèmes est décidable pour les grammaires linéaires à droite (hors-contexte, etc.) s'il existe un algorithme qui pour toute grammaire linéaire à droite (hors-contexte, etc.) calcule la réponse (oui ou non) au problème posé ; il est dit indécidable dans le cas contraire. Par exemple tous les problèmes énoncés ci-dessus sont décidables pour les grammaires régulières (la réponse au quatrième problème est toujours oui) ; les problèmes du langage vide et du langage infini sont décidables pour les grammaires hors-contexte (voir les chapitres IX et VIII) ; les deux autres problèmes sont indécidables pour cette classe de grammaires.
D'autres problèmes importants sont :
– le problème du mot : étant donné une grammaire G et une chaîne terminale x, est-ce que x appartient à $L(G)$?
– le problème de l'équivalence : étant données deux grammaires G_1 et G_2, est-ce que $L(G_1) = L(G_2)$?
Le problème du mot est décidable pour les grammaires sous-contexte (voir l'exercice 3.1 ci-dessus), pour les grammaires hors-contexte (voir le chapitre III) et donc pour les grammaires linéaires à droite qui sont des grammaires hors-contexte particulières ; il est indécidable pour les grammaires générales . Si le problème du mot est décidable pour une classe de grammaires, les langages engendrés par les grammaires de cette classe sont récursifs. Le problème de l'équivalence est indécidable pour les grammaires hors-contexte ; il est décidable pour les grammaires linéaires à droite (voir le chapitre VI).

3.3.7 Choix d'un formalisme (classe de définitions) pour définir un langage de programmation

Les types de grammaires introduits au paragraphe 3.3 constituent différents formalismes que l'on peut utiliser pour définir la syntaxe d'un langage de programmation. Pour choisir parmi ces formalismes on est guidé par les critères suivants :

1. Le formalisme doit être suffisamment puissant pour contenir une définition du langage.
2. Le formalisme doit être suffisamment simple pour que chaque définition puisse être compilée en un algorithme efficace de reconnaissance des programmes du langage défini.

3. Le formalisme doit contenir une définition du langage concise et facile à comprendre.

Le critère 1) conduit à exclure le formalisme des grammaires régulières, qui ne permet pas de décrire les parenthésages à une profondeur arbitrairement grande que l'on rencontre dans les langages de programmation (expressions avec parenthèses, structure de bloc, etc.). Les grammaires régulières (ou des formalismes équivalents) sont par contre souvent utilisées pour définir la lexicographie des langages de programmation (c'est-à-dire les identificateurs et les constantes).
Le critère 2) conduit à exclure les grammaires générales (qui peuvent engendrer des langages non récursifs) et les grammaires sous-contexte, auxquelles on ne peut pas associer des algorithmes de reconnaissance efficaces.
C'est pourquoi la plupart des langages de programmation sont définis par des grammaires hors-contexte (ou des formalismes équivalents comme celui des diagrammes syntaxiques). Le critère 1) est alors à peu près satisfait (mais pas complétement ; par exemple le traitement des déclarations dans un langage de programmation ne peut être décrit par une grammaire hors-contexte, comme on le verra au chapitre VIII). Le critère 2) est lui aussi à peu près satisfait (on utilise souvent des grammaires hors-contexte particulières admettant des algorithmes de reconnaissance plus efficaces, comme les grammaires LL(1) ou LR(1)). Enfin définir un langage avec une grammaire hors-contexte revient à en donner une définition par récurrence qui satisfait souvent le critère 3) (voir le chapitre IX).

3.4 Exemples

3.4.1 Définition des nombres réels sans signe d'un langage de programmation avec une grammaire hors-contexte

- $V_T = \{0,1,2,3,4,5,6,7,8,9,+,-,E,.\}$
- $V_N = \{R,D,N,F\}$
- Axiome : R
- Règles :

$$\begin{array}{lcl} R & \rightarrow & D|DF|NF \\ D & \rightarrow & N.N \\ F & \rightarrow & EN|ESN \\ N & \rightarrow & C|NC \\ S & \rightarrow & +|- \\ S & \rightarrow & 0|1|2|3|4|5|6|7|8|9 \end{array}$$

Exercice 3.2

1. *Montrer que* $R \Rightarrow^* 2.3E-6$
2. *Trouver une grammaire régulière équivalente à la grammaire donnée.*

3.4.2 Langage $\{a^n b^n | n \geq 1\}$

Soit la grammaire hors-contexte $G_1 =< \{a,b\},\{S\},S,\{S \rightarrow aSb|ab\} >$.
Il est facile de montrer que $L(G_1) = \{a^n b^n | n \geq 1\}$.

3.4.3 Langage des parenthèses des expressions

Soit la grammaire hors-contexte $G_2 =< \{a,b\},\{S\},S,\{S \rightarrow aSb|SS|\varepsilon\} >$. Les symboles a et b représentant respectivement le début et la fin d'un bloc, le langage engendré par G_2 décrit toutes les structures de blocs possibles. La grammaire G_2 est équivalente à la grammaire $G_3 =< \{a,b\},\{S\},S,\{S \rightarrow SaSb|\varepsilon\} >$
et à la grammaire $G4 =< \{a,b\},\{S\},S,\{S \rightarrow aSbS|\varepsilon\} >$.

3.4.4 Grammaire engendrant les expressions régulières sur {a, b}

- $V_T = \{a,b,\emptyset,+,.,*,(,)\}$
- $V_N = \{E,T,F,X\}$
- Axiome : E

– Règles :

$$\begin{array}{lcl} E & \rightarrow & E+T|T \\ T & \rightarrow & T.F|F \\ F & \rightarrow & X^*|X \\ X & \rightarrow & \emptyset|a|b|(E) \end{array}$$

3.4.5 Langage $\{a^n b^n c^n | n \geq 1\}$

Soit la grammaire $G_5 =< \{a,b,c\},\{S,B,C\},S,R >$, où l'ensemble des règles R est :

$$\begin{array}{lcl} S & \rightarrow & aSBC|aBC \\ CB & \rightarrow & BC \\ aB & \rightarrow & ab \\ bB & \rightarrow & bb \\ bC & \rightarrow & bc \\ cC & \rightarrow & cc \end{array}$$

L'ensemble R satisfait la contrainte suivante : pour tout $\alpha \rightarrow \beta$ appartenant à R, $0 < |\alpha| \leq |\beta|$. On peut montrer que les grammaires satisfaisant cette contrainte sont équivalentes aux grammaires sous-contexte définies au paragraphe 3.3.2 (Exercice 3.8). La grammaire G_5 engendre le langage $\{a^n b^n c^n | n \geq 1\}$. On montrera au chapitre VIII que ce langage n'est pas hors-contexte.

3.5 Exercices

Exercice 3.3 *Construire des grammaires hors-contexte pour les langages suivants*

1. $\{ww^{\sim} | w \in \{a,b\}^*\}$
2. $\{a^i b^j | i \neq j \text{ et } i,j \geq 0\}$

Exercice 3.4

1. *On définit l'ensemble des formules de la logique propositionnelle comme l'ensemble des expressions construites avec l'opérateur $\neg$ (non), les opérateurs binaires $\vee$ (ou), $\wedge$ (et), $\rightarrow$ (implique), et des opérandes qui sont des symboles de propositions atomiques. On utilise l'ensemble des suites de lettres sur l'alphabet $\{A,B,\ldots,Z\}$ pour noter les propositions atomiques.*
   Exemples : $\neg A \vee B$, $A \rightarrow (B \rightarrow A)$, $AA \wedge B \vee \neg AA \wedge C$, $(ABA \rightarrow BB) \wedge (ABA \rightarrow \neg C)$, ... sont des formules de la logique propositionnelle.
   Donner une grammaire hors-contexte engendrant l'ensemble des formules de la logique propositionnelle.
2. *Soit $F = \{a,b,f,g,h\}$, où a et b sont deux symboles de constantes, f et g deux symboles de fonctions unaires, h un symbole de fonction binaire. L'ensemble des termes sur F est l'ensemble des expressions que l'on peut construire avec les éléments de F.*
   Exemple : $h(a,b), f(f(b)), h(f(a),h(b,b)), \ldots$ sont des termes.
   Donner une grammaire hors-contexte engendrant l'ensemble des termes sur F.
3. *Soit $G = \{=,<\}$, où $=$ et $<$ sont deux symboles de prédicats binaires. L'ensemble des propositions atomiques sur $F \cup G$ est l'ensemble des expressions de la forme $t_1 = t_2$ ou $t_1 < t_2$, où t_1 et t_2 sont des termes sur F.*
   Donner une grammaire hors-contexte engendrant l'ensemble des propositions atomiques sur $F \cup G$.
4. *Soit X un ensemble infini de symboles de variables. On pose par exemple $X = \{x,y,z\}^+$.*
   L'ensemble des termes *sur $F \cup X$, avec F défini en 2, est alors l'ensemble des expressions construites avec des opérandes pris dans $\{a,b\} \cup X$, et des opérateurs pris dans $\{f,g,h\}$.*
   L'ensemble des formules atomiques *sur $F \cup G \cup X$ est l'ensemble des expressions de la forme $t1 = t2$ ou $t1 < t2$, où $t1$ et $t2$ sont des termes sur $F \cup X$.*
   L'ensemble des formules de la logique des prédicats *sur $F \cup G \cup X$ est l'ensemble des expressions construites avec les formules atomiques sur $F \cup G \cup X$, les opérateurs de la logique propositionnelle $\neg, \vee, \wedge, \rightarrow$ et les quantificateurs $\forall$ et $\exists$.*

Exemples : $(\forall x f(x) = f(x))$, $(\exists x f(x) = g(x) \rightarrow (\forall x f(x) = g(x)))$, $(\forall x f(x) = f(y))$, $f(x) < g(y) \vee (\exists z f(z) < f(x))$, $(\forall x \forall y \forall z x < y \wedge y < z \rightarrow x < z)$, ... *sont des formules de la logique des prédicats.*

Donner une grammaire hors-contexte engendrant l'ensemble des formules de la logique des prédicats.

Exercice 3.5
Soit $G =< V_T, V_N, S, R >$ *la grammaire définie par :*
$V_T = \{a,b,c\}$, $V_N = \{S,A,B\}$, *R est l'ensemble de règles*

$$\begin{array}{lcl} S & \rightarrow & aSA|bSB|c \\ cA & \rightarrow & ca \\ cB & \rightarrow & cb \\ aA & \rightarrow & Aa \\ aB & \rightarrow & Ba \\ bA & \rightarrow & Ab \\ bB & \rightarrow & Bb \end{array}$$

1. *Quel est le type de cette grammaire ?*
2. *Donner une dérivation de la chaîne abcab.*
3. *Prouver que* $L(G) = \{wcw \mid w \in \{a,b\}*\}$.

Exercice 3.6 *Décrire les langages engendrés par les grammaires suivantes, où S est l'axiome et les non-terminaux sont les lettres majuscules apparaissant dans les règles :*

1. *Grammaire hors-contexte :*
   $S \rightarrow bSS|a$
2. *Grammaire sous-contexte :*

$$\begin{array}{lcl} S & \rightarrow & \#a\# \\ \#a & \rightarrow & \#B \\ Ba & \rightarrow & aaB \\ B\# & \rightarrow & aa\# \end{array}$$

Exercice 3.7 *Soit* $G =< V_T, V_N, S, R >$ *une grammaire. Montrer qu'il existe une grammaire équivalente* $G' =< V_T, V'_N, S', R' >$ *dont toutes les règles* $\alpha \rightarrow \beta$ *sont telles que* $\alpha \in V^+_{N'}$.

Exercice 3.8 *Montrer qu'un langage est sous-contexte si et seulement si il est engendré par une grammaire dont toutes les règles* $\alpha \rightarrow \beta$ *sont telles que* $0 < |\alpha| \leq |\beta|$.

Exercice 3.9 *Montrer qu'un langage ne contenant pas la chaîne vide est engendré par une grammaire linéaire à droite si et seulement si il est engendré par une grammaire dont toutes les règles sont de l'une des deux formes suivantes :*

- $A \rightarrow aB$, *avec* $A, B \in V_N$, $a \in V_T$
- $A \rightarrow a$, *avec* $A \in V_N$, $a \in V_T$

Exercice 3.10 *Montrer qu'il n'existe pas de grammaire linéaire à droite engendrant le langage* $\{a^n b^n | n \geq 1\}$.

Chapitre III

Grammaires hors-contexte

1 Propriété fondamentale (décomposition des dérivations hors-contexte)

La propriété 2.4 du chapitre II signifie que l'on peut composer des dérivations d'un système de réécriture ; la réciproque de cette propriété, qui est fausse pour les systèmes de réécriture généraux, est vraie pour les grammaires hors-contexte : on peut décomposer une dérivation hors-contexte en sous-dérivations indépendantes. Cette propriété fondamentale des dérivations hors-contexte est très souvent utilisée dans les démonstrations.

Théorème 1.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. On pose $V = V_T \cup V_N$.*
Soit une dérivation $u_1 \dots u_n \Rightarrow^p w$, avec $u_i \in V^*$ pour $i = 1, \dots, n$ et $w \in V^*$.
Alors pour $i = 1, \dots, n$, il existe $v_i \in V^*$ et $q_i \in \mathbb{N}$ tels que :
- *$w = v_1, \dots, v_n$*
- *pour $i = 1, \dots, n, u_i \Rightarrow^{q_i} v_i$*
- *$p = \sum_{i=1}^{n} q_i$*

Preuve : On procède par récurrence sur p.
- Si $p = 0$, le théorème est vrai.
- Supposons que le théorème soit vrai pour toute dérivation de longueur p.
  Considérons une dérivation $u_1 \dots u_n \Rightarrow^{p+1} w$. Puisque les parties gauches des règles d'une grammaire hors-contexte sont de longueur 1, la première règle de la dérivation considérée est appliquée à un u_i. Il existe donc u_i et $t_i \in V^*$ tels que :
  $u_1 u_{i-1} u_i u_{i+1} \dots u_n \Rightarrow u_1 \dots u_{i-1} t_i u_{i+1} \dots u_n \Rightarrow_p w$.
  On applique l'hypothèse de récurrence à la sous-dérivation de longueur p : il existe $v_1, \dots, v_n, q_1, \dots, q_n$ tels que :
  . $w = v_1 \dots v_n$
  . pour k de 1 à $i-1$ et de $i+1$ à n, $u_k \Rightarrow^{q_k} v_k$
  . $t_i \Rightarrow^{q_i} v_i$
  . $p = \sum_{j=1}^{n} q_j$
  Puisque $u_i \Rightarrow^{q_i+1} v_i$, le théorème est vérifié pour la dérivation $u_1 \dots u_n \Rightarrow^{p+1} w$.

□

La propriété fondamentale a été énoncée pour la relation de dérivation Δ_2 (c'est-à-dire la relation $\Rightarrow^p$). Nous énonçons ci-dessous comme corollaire la propriété fondamentale pour la relation $\Rightarrow^*$ (relation de dérivation Δ_1).

Corollaire 1.1 *Si $u_1 \dots u_n \Rightarrow^* w$ dans une grammaire hors-contexte, alors il existe $v_1, \dots, v_n$ tels que $w = v_1 \dots v_n$ et $u_1 \Rightarrow^* v_1, \dots, u_n \Rightarrow^* v_n$.*

Nous étudierons au paragraphe 2 la propriété fondamentale pour les relations de dérivation Δ_3 et Δ_4.

Remarque 1.1 *Dans la suite de ce chapitre, sauf exceptions signalées, nous ne parlerons que des grammaires hors-contextes en utilisant implicitement les notations du théorème ci-dessus.*

Exercice 1.1

1. *Donner un contre-exemple montrant que la propriété fondamentale est fausse pour une grammaire qui n'est pas hors-contexte.*
2. *Soit $< V, R >$ un système de réécriture tel que toutes ses dérivations satisfont la propriété fondamentale. Prouver que toutes ses règles sont de la forme $a \rightarrow \omega$, avec a appartenant à V et ω appartenant à V^*. Ainsi la propriété fondamentale caractérise les systèmes de réécriture dont toutes les règles ont une partie gauche de longueur 1.*

2 Dérivations indicées dans une grammaire hors-contexte

La notion de dérivation indicée a été définie au chapitre II, paragraphe 2.2. Rappelons que si $x \Rightarrow_R y$ dans un système de réécriture R, il est possible que

1. plusieurs règles de R permettent de réécrire x en y
2. une même règle de R puisse s'appliquer à différentes sous-chaînes de x pour donner y.

On a levé ces ambiguïtés en utilisant une relation $\Rightarrow_{i,r}$ où i est le rang du premier caractère de la sous-chaîne de x qui est réécrite, et r est la règle utilisée.
Quand R est un ensemble de règles hors-contexte il suffit d'indiquer le rang du caractère de x qui est dérivé : la donnée de la chaîne de départ x, de la chaîne obtenue y et de l'indice i du caractère réécrit permet de retrouver la règle r appliquée.
Aussi nous simplifions les notations introduites au paragraphe 2.2, en ne précisant dans les dérivations indicées, que la position *du* symbole dérivé.

2.1 Dérivations indicées gauches

Définition 2.1 (Dérivation indicée gauche) *Soit $x_1, \ldots x_k \in V^*$ et $i_1, \ldots i_{k-1} \in \mathbb{N}$.*
Une dérivation indicée dans une grammaire hors-contexte $x_1, i_1, \ldots, x_{k-1}, i_{k-1}, x_k$, est dite dérivation indicée gauche *si les positions des symboles dérivés sont croissantes, c'est-à-dire si $i_j \leq i_{j+1}$ pour j de 1 à $k-2$.*

Théorème 2.1 *Pour toute dérivation indicée, il y a une dérivation indicée gauche de même longueur, origine et extrémité*

Preuve :
Soit $D = x_1, i_1, \ldots, i_{k-1}, x_k$ une dérivation indicée où $x_1, x_2 \ldots, x_k \in V^*$ et $i_1, \ldots i_{k-1} \in \mathbb{N}$.
Soit $P(n)$ la propriété suivante :
- n est le nombre d'inversions de la suite des indices
- il y a une dérivation indicée gauche de même longueur, origine et extrémité que la dérivation ci-dessus.

$P(0)$ est trivialement vraie.
Soit $n > 0$. Supposons $P(j)$ vraie pour tout $j < n$ et prouvons $P(n)$. Puisque $n > 0$, il existe p tel que $p \leq k-2$ et $i_p > i_{p+1}$. Par suite
- $x_p = uAvBw$ où $u, v, w \in V^*, A, B \in V_N$ et $i_p = |uAvB|, i_{p+1} = |uA|$
- $x_{p+1} = uAv\beta w$ où $B \rightarrow \beta$
- $x_{p+2} = u\alpha v\beta w$ où $A \rightarrow \alpha$

Dans la dérivation D, on remplace $x_p, i_p, x_{p+1}, i_{p+1}, x_{p+2}$ par $x_p, i_{p+1}, y, j, x_{p+2}$ où
$y = u\alpha vBw$ et $j = |u\alpha vB| = |uAvB| + |\alpha| - 1 = i_p + |\alpha| - 1$.
Puisque $i_{p+1} < i_p$, on a $i_{p+1} \leq j$.
Soit D' la nouvelle dérivation ainsi construite (par permutation des applications de deux règles). Cette dérivation comporte moins de n inversions, donc par hypothèse de récurrence, il existe dérivation indicée gauche de même longueur, origine et extrémité que la dérivation D' donc de D. $\square$

Remarque 2.1 *En analysant la preuve ci-dessus, on constate que la nouvelle dérivation ne fait que permuter l'ordre d'application des règles.*

Exemple 2.1 *Soit $A \rightarrow AA|a$ une grammaire hors-contexte et la dérivation indicée :*
$AA \Rightarrow_2 AAA \Rightarrow_1 AAAA$.
En permutant l'ordre d'application de la règle $A \rightarrow AA$, on obtient la dérivation indicée gauche :
$AA \Rightarrow_1 AAA \Rightarrow_3 AAAA$.

2.2 Dérivations gauches

Définition 2.2 *Soit $x_1 \ldots x_k$ une dérivation (définition $\Delta 3$ - cf le paragraphe 2.2 du chapitre II) dans une grammaire hors-contexte. On dit que cette dérivation est une* dérivation gauche *si pour chaque élément x_i de la dérivation le non-terminal réécrit est le non-terminal le plus à gauche de x_i.*
Pour une grammaire hors-contexte donnée, on définit la relation $\Rightarrow_g$ par :
$x \Rightarrow_g y$ si et seulement si il existe une chaîne terminale u, un non-terminal A, une chaîne v et une règle $A \rightarrow \alpha$ tels que $x = uAv$ et $y = u\alpha v$.
Une dérivation $x_1, \ldots x_k$ est une dérivation gauche si et seulement si $x_i \Rightarrow_g x_{i+1}$ pour $i = 1 \ldots k-1$.

Exemple 2.2 *Soit la grammaire suivante avec $V_T = \{a, +, *, (,)\}, V_N = \{E, T, F\}$ et les règles :*

$$\begin{array}{lcl} E & \rightarrow & E+T|T \\ T & \rightarrow & T*F|F \\ F & \rightarrow & (E)|a \end{array}$$

On a : $E \Rightarrow_g E+T \Rightarrow_g T+T \Rightarrow_g F+T \Rightarrow_g a+T \Rightarrow_g a+F \Rightarrow_g a+a$.

En fait une dérivation gauche est un cas particulier d'une dérivation indicée gauche, comme le montre la preuve du théorème suivant.

Théorème 2.2 *Soit A un non-terminal et x une chaîne terminale. Si $A \Rightarrow^p x$, alors $A \Rightarrow_g^p x$.*

Preuve : Si $A \Rightarrow^p x$, il existe, d'après le théorème 2.1, une dérivation indicée gauche
$x_1, i_1, \ldots x_{k-1}, i_{k-1}, , x_k$ avec $x_1 = A$ et $x_k = x$.
La dérivation $D = x_1, x_2, \ldots x_k$ est alors une dérivation gauche d'origine A et d'extrémité x. En effet, soit $x_j \neq x$ un élément de cette dérivation. Aucune lettre de x_j d'indice strictement inférieur à i_j (c'est-à-dire à gauche du symbole réécrit de x_j) n'est réécrite dans la dérivation D. Toutes ces lettres sont donc des lettres de x, et la chaîne x étant terminale, sont donc des lettres terminales. Il s'ensuit que D est une dérivation gauche. □

Remarque 2.2 *On peut définir de façon analogue des* dérivations indicées droites *et des* dérivations droites*, et démontrer les équivalents pour ces dérivations des théorèmes 2.1 et 2.2*

3 Décomposition des dérivations indicées gauches

Définition 3.1 (Somme de deux dérivations indicées gauches) *Soit D_1 la dérivation indicée gauche*
$x_1, i_1, \ldots, x_{k-1}, i_{k-1}, x_k$ *où* $x_1, \ldots x_k \in V^*, i_1, \ldots i_{k-1} \in \mathbb{N}$.
Soit D_2 la dérivation indicée gauche
$y_1, j_1, \ldots, y_{l-1}, j_{l-1}, y_l$ *où* $y_1, \ldots y_l \in V^*, j_1, \ldots j_{l-1} \in \mathbb{N}$.
$D_1 + D_2$ est la dérivation indicée d'origine $x_1 y_1$ obtenue en effectuant D_1 puis D_2, c'est-à-dire la dérivation :
$x_1 y_1, i_1, \ldots x_{k-1} y_1, i_{k-1}, x_k y_1, |x_k| + j_1, \ldots, x_k y_{l-1}, |x_k| + j_{l-1}, x_k y_l$.
Il est clair que, puisque D_1 et D_2 sont des dérivations indicées gauches, il en est de même de $D_1 + D_2$

Remarque 3.1 *Il est facile de voir que cette somme est associative, ce qui rend sans amiguïté la somme de n dérivations $D_1 + D_2 + \ldots + D_n$.*

Exemple 3.1 *Soit la grammaire* $S \rightarrow \varepsilon|aSbS|bSaS$.
Soit D la dérivation indicée $S,1,\varepsilon$.
Notons que a (respectivement b) sont des dérivations indicées gauches de longueur 0.
$b+D = bS,2,b$
$a+D+b = bSa,2,ba$
$a+D+b+D = bSaS,2,baS,3,ba$

Propriété 3.1 (Décomposition des dérivations indicées gauches) *Soit* $x = u_1u_2$ *où* $u_1,u_2 \in V^*$ *et D une dérivation indicée gauche d'origine x. Il existe une et une seule décomposition de D en deux dérivations gauches* D_1 *d'origine* u_1 *et* D_2 *d'origine* u_2 *telles que* $D = D_1 + D_2$.

Preuve : Soit $D = w_1,i_1,\ldots w_{k-1},i_{k-1},w_k$ une dérivation indicée gauche où $x = w_1$, $w_1,\ldots,w_k \in V^*$, $i_1,\ldots i_{k-1} \in \mathbb{N}$.
Si $k = 0$ alors $D_1 = u_1$ et $D_2 = u_2$.
Supposons $k > 0$ et la propriété vraie pour toute dérivation indicée gauche de longueur inférieure à k.
Montrons que la propriété est aussi vérifiée par D.
Considérons les deux cas suivants :

1. $i_1 > |u_1|$.
   Dans ce cas $D_1 = u_1$.
   Pour n de 1 à k, soit v_n tel que $w_n = u_1v_n$ (v_n est le suffixe de u_1 dans w_n).
   Pour n de 1 à k, soit $j_n = i_n - |u_1|$.
   On a $D_2 = v_1,j_1,\ldots,j_{k-1},v_k$
2. $i_1 \leq |u_1|$.
   Donc il y a des chaines s,t,α et un non terminal A tels que $w_1 = sAtv_1$, $A \rightarrow \alpha$, $w_2 = s\alpha tv_1$.
   Soit D' la partie de la dérivation D commençant en w_2. Vu l'hypothèse de récurrence, il y une décomposition *unique* de D' en D'_1 d'origine $s\alpha t$ et D_2 d'origine v_1.
   Soit $D_1 = s\alpha t,i_1,D'_1$ (ce qui assure l'unicité de D_1). On a : $D = D_1 + D_2$.

□

Propriété 3.2 (Décomposition des dérivations indicées gauches) *Soit* $x = u_1u_2,\ldots,u_n$ *où* $u_1,u_2,\ldots,u_n \in V^*$ *et D une dérivation indicée gauche d'origine x. Il existe une et une seule décomposition de D en n dérivations indicées gauches* D_1 *d'origine* u_1 *et* D_2 *d'origine* u_2, $\ldots D_n$ *d'origine* u_n *telles que* $D = D_1 + D_2 + \ldots + D_n$.

Preuve : La preuve est une généralisation par récurrence sur n de la propriété 3.1 ci-dessus. □

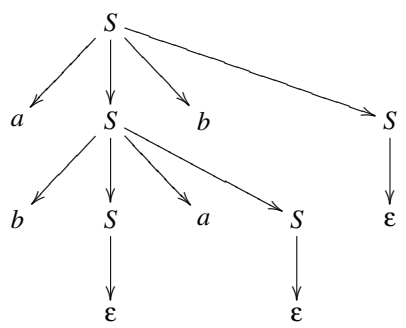
4 Arbre de dérivation

Les arbres de dérivation sont une autre représentation plus visuelle du processus de dérivation. Nous verrons qu'il y a une bijection entre les arbres de dérivations et les dérivations indicées gauche ayant comme origine un symbole. Nous donnerons deux définitions des arbres de dérivation, la première définit un arbre comme une expression et la deuxième est plus proche des arbres utilisés en informatique. Pour démontrer l'existence de la bijection, nous utiliserons la représentation des arbres dérivation par des expressions, mais en pratique, on préfère la deuxième représentation.
Dans ce paragraphe, on note les grammaires comme précédemment par $G =< V_T,V_N,S,R >$ et on pose $V = V_T \cup V_N$.

Exemple 4.1 *Soit la grammaire* $S \rightarrow aSbS|bSaS|\varepsilon$ *et la dérivation indicée gauche :*
$S \Rightarrow_1 aSbS \Rightarrow_2 abS \Rightarrow_3 abaSbS \Rightarrow_4 ababS \Rightarrow_5 abab$.
Les indices sont inutiles car cette dérivation est une dérivation gauche où le non-terminal le plus à gauche est dérivé à chaque pas de la dérivation.
L'arbre de dérivation est représenté,

- *soit par l'expression* $S(a,S(b,S(),a,S()),b,S())$

– *soit par l'arbre*



Définition 4.1 (Arbre = Expression) *Un arbre de dérivation d'origine a où $a \in V$ pour la grammaire G est*

– *soit a*
– *soit $a(e_1, \ldots, e_n)$ où : $n \geq 0$*
  Pour i de 1 à n, e_i est un arbre de dérivation d'origine b_i
  $a \rightarrow b_1 \ldots b_n$ est une règle de la grammaire G

Exemple 4.2 *Pour la grammaire $S \rightarrow aSbS|bSaS|\varepsilon$ de l'exemple précédent, les expressions $a, b, S(), S(b, S(), a, S()), S(a, S(b, S(), a, S()), b, S())$ sont des arbres de dérivation.*

Définition 4.2 (Arbre = Domaine d'arbre et étiquettes)

1. *Domaine d'arbre : un* domaine d'arbre *est un ensemble de suites finies d'entiers naturels*
   – *qui est fermé par préfixe : si $u.v \in D$ alors $u \in D$*
   – *qui vérifie : si $i \in \mathbb{N}$ et $u.(i+1) \in D$ alors $u.i \in D$*
   On reconnaîtra dans un domaine d'arbre la numérotation usuelle des livres (dont cet ouvrage) : si le paragraphe 2.3.2 *existe, alors existent notamment les paragraphes* 2 *et* 2.3.1.
2. *Arbre : soit D un domaine d'arbre fini, un* arbre de dérivation *pour la grammaire G est une application a de D dans $V \cup \{\varepsilon\}$ telle que :*
   si $n \in \mathbb{N}$, $u \in D$, $u.n \in D$ et $u.(n+1) \notin D$ alors
   $a(u) \rightarrow a(u1) \ldots a(un)$ est une règle de G

Soit $a : D \mapsto V$ un arbre de dérivation :

– *$u \in D$ est un* sommet *de l'arbre.*
– *ε (la suite vide d'entiers) est la* racine *de l'arbre.*
– *$a(u)$ est* l'étiquette *du sommet u.*
– *Si $u \in D$, $i \in \mathbb{N}$ et $u.i \in D$ alors le sommet ui est un* successeur *du sommet u. item Si $u \in D$ et u n'a pas de successeur, alors u est une* feuille *de l'arbre.*
– *Si $u \in D$ et $uv \in D$, alors uv est un* descendant *du sommet u.*

Les arbres de dérivation ainsi définis, sont dessinés comme dans l'exemple 4.1. Les suites d'entiers sont omises et remplacées par des flêches vers le bas allant d'un sommet vers chacun de ses successeurs. Le successeur $u.(i+1)$ est placé à droite du sommet $u.i$.

Théorème 4.1 *Soit G une grammaire. Il y a une bijection entre les arbres de dérivation de G et les dérivations indicées gauches ayant pour origine un symbole de G.*

Preuve : Soit V le vocabulaire de G. On définit une application $a2d$ entre les arbres de dérivation de G (suivant la définition 4.1) et les dérivations indicées gauches ayant pour origine un symbole de G par :

– $a2d(a) = a$ où $a \in V$
– $a2d(a(e_1, \ldots e_n)) = a, 1, a2d(e_1) + \ldots + a2d(e_n)$

Nous prouvons que $a2d$ est une bijection.

1. $a2d$ est injective.
   Soient deux arbres T et U tels que $a2d(T) = a2d(U)$. Par récurrence sur les sous-arbres de T, on montre que $T = U$.
   Admettons que la propriété ci-dessus soit vérifiée pour tout sous-arbre strict de T.
   (a) $T \in V$.
   Par définition de $a2d$, $a2d(T) = T$. Puisque $a2d(U) \in V$, seul le premier cas de la définition de $a2d$ s'applique, donc $a2d(U) = U = T$.
   (b) $T = a(e_1, \ldots, e_n)$.
   $a2d(T) = a, 1, a2d(e_1) + \ldots + a2d(e_n)$.
   Soit D la dérivation indicée gauche $a2d(e_1) + \ldots + a2d(e_n)$.
   Pour i de 1 à n, soit b_i l'origine de l'arbre e_i.

      i. Puisque $a2d(T) = a2d(U)$ alors U et T ont même origine, donc $U = a(f_1, \ldots, f_p)$.
      Pour i de 1 à p, soit c_i la racine de f_i.
      ii. Puisque $a2d(T) = a2d(U)$ et que donc $b_1 \ldots b_n$ et $c_1 \ldots c_p$ sont l'origine de la dérivation D, il en résulte que $n = p$ et pour i de 1 à n, $b_i = c_i$ et $U = a(f_1, \ldots, f_n)$.
      iii. Par définition de $a2d$, $a2d(U) = a, 1, a2d(f_1) + \ldots + a2d(f_n)$ où $D = a2d(f_1) + \ldots + a2d(f_n)$.
      En résumé $D = a2d(e_1) + \ldots + a2d(e_n) = a2d(f_1) + \ldots + a2d(f_n)$, D a comme origine $b_1 \ldots b_n$ et pour i de 1 à n, b_i est la racine commune à $a2d(e_i)$ et $a2d(f_i)$.
      D'après la propriété 3.2 il en résulte que pour i de 1 à n, $a2d(e_i) = a2d(f_i)$.
      Par hypothèse de récurrence pour i de 1 à n, $e_i = f_i$, donc $T = U$.
2. $a2d$ est surjective.
   On le prouve par récurrence sur la longueur des dérivations.
   Soit D une dérivation indicée gauche ayant comme origine le symbole $a \in V$.
   Si D est de longueur O, alors $D = a$ et $a2d(D) = D$.
   Supposons que D soit de longueur non nulle donc de la forme $D = a, 1, D'$.
   Soit $b_1 \ldots b_n$ l'origine de D' où pour i de 1 à n, b_i est un symbole.
   Puique D est une dérivation, $a \rightarrow b_1 \ldots b_n$ est une règle de la grammaire.
   D'après la propriété 3.2, $D' = D_1 + \ldots + D_n$ où
   pour i de 1 à n, D_i est une dérivation indicée gauche d'origine b_i.
   Par hypothèse de récurrence,pour i de 1 à n, il existe un arbre de dérivation e_i d'origine b_i tel que $a2d(e_i) = D_i$.
   Par la définition 4.1 des arbres, $a(e_1, \ldots, e_n)$ est un arbre de dérivation de la grammaire G.
   Par définition de $a2d$, $a2d(a(e_1, \ldots, e_n)) = D$. Donc $a2d$ est surjective.

□

5 Ambiguïté

Définition 5.1 *Soit $G = V_T, V_N, S, R >$ une grammaire hors contexte.*
La grammaire G est ambigüe s'il existe une chaîne terminale admettant au moins deux arbres de dérivation de racine S, soit encore de façon équivalente, d'après le théorème 4.1, deux dérivations gauches d'origine S.
Une chaîne terminale qui est l'extrémité d'au moins deux dérivations gauches d'origine S est une chaîne ambigüe de la grammaire G.
Un langage hors-contexte est ambigu si toutes *les grammaires hors-contextes qui l'engendrent sont ambigües.*

Exercice 5.1 *Soit la grammaire suivante, ayant pour vocabulaire terminal l'unique symbole a, et comme règles :*
$S \rightarrow A|B, A \rightarrow a, B \rightarrow a$

1. *Montrez qu'elle est ambigüe.*
2. *Le langage engendré par cette grammaire n'est pas ambigu. Donnez une grammaire non ambigüe équivalente.*

Exercice 5.2 *Soit la grammaire G suivante, de vocabulaire terminal* $0, 1$, *et de règles* $S \rightarrow 0S1|0S11|\varepsilon$

1. *Montrez qu'elle est ambigüe.*
2. *Prouvez que* $L(G) = \{O^n 1^m \mid n \leq m \leq 2n\}$.
3. *Montrez que le nombre de dérivations gauches dans la grammaire G de la chaîne* $O^n 1^m$ *est* $\binom{n}{m-n}$.
   Indication *: la longueur de la dérivation de cette chaine est* $n+1$. *Une dérivation particulière est caractérisée par le choix des* $m-n$ *emplacements de l'usage de la règle* $S \to OS11$.
4. *Donnez une grammaire non ambigüe équivalente.*

Exercice 5.3 *La grammaire suivante, de vocabulaire terminal* a, b, *et de règles* $S \to aSb|SS|\varepsilon$ *peut être vue comme décrivant la structure des parenthèses d'une expression bien parenthésée, a jouant le rôle de la parenthèse ouvrante et b celui de la parenthèse fermante.*

1. *Montrez que cette grammaire est ambigüe.*
2. *Montrez qu'elle est équivalente à la grammaire* $S \to aSbS|\varepsilon$
3. *Montrez que la grammaire ci-dessus n'est pas ambigüe*

Exemple 5.1 *Le langage* $\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$ *est un langage hors-contexte ambigu. Il est facile de donner une grammaire hors-contexte qui l'engendre. Il est intuitif mais difficile de prouver que ce langage est ambigu, c'est pourquoi nous le proposons pas en exercice.*

Remarque 5.1 ***Il n'y a pas d'algorithme*** *permettant de répondre à la question "la grammaire G est-elle ambigüe ?" : le problème de l'ambigüité d'une grammaire est indécidable.*

Exercice 5.4 *Soit la grammaire suivante, de vocabulaire terminal* `si`, `alors`, `sinon`, e, a, *et de règles*
$I \to$ `si` e `alors` $I|$ `si` e `alors` I `sinon` $I|a$.

1. *Montrez que cette grammaire est ambigüe.*
2. *Trouvez une grammaire équivalente non ambigüe.*

Exercice 5.5 *Soit la grammaire G suivante, de vocabulaire terminal* a, b, c *et de règles*
$S \to aSc|aS|T$
$T \to bTc|bT|\varepsilon$

1. *Montrez que cette grammaire est ambigüe.*
2. *Montrez que* $L(G) = \{a^p b^q c^r \mid p + q \geq r \geq 0\}$
3. *Trouvez une grammaire équivalente non ambigüe.*

Exercice 5.6 *Soit* $L = \{w \in \{a,b\}^* \mid w \text{ contient autant de } a \text{ que de } b\}$. *Soient les trois grammaires* G_1, G_2, G_3 *ayant pour règles*

- G_1 *:* $S \to \varepsilon|aSb|bSa|SS$
- G_2 *:* $S \to aSbS|bSaS|\varepsilon$
- G_3 *:* $S \to aB|bA|\varepsilon$
  $A \to aS|bAA$
  $B \to bS|aBB$

1. *Montrez que ces trois grammaires engendrent le langage L.*
2. *Montrez qu'elles sont toutes ambigües*
3. *Donnez une grammaire non ambigüe engendrant L*

Exercice 5.7 *Soit* $V_T = \{0,1\}$ *un vocabulaire terminal et* $L = V_T^* - \{ww \mid w \in V_T^*\}$. *Montrez que L est un langage hors-contexte. Le problème n'est pas facile et nous le décomposons en questions intermédiaires. Il est clair que les chaînes de longueur impaire (sur le vocabulaire terminal* $\{0,1\}$*) sont éléments de L.*

1. *Soit Y l'ensemble des chaînes de longueur impaire dont le milieu est 1.*
   Trouvez une grammaire engendrant Y.
   Soit Z l'ensemble des chaînes de longueur impaire dont le milieu est 0.
   Trouvez une grammaire engendrant Z.

2. *Montrez que toute chaîne de $YZ \cup ZY$ n'est pas de la forme ww*
3. *Montrez que toute chaîne de longueur paire, qui n'est pas de la forme ww, est élément de $YZ \cup ZY$.*
4. *En déduire une grammaire engendrant L*

On trouvera un excellent corrigé de cet exercice dans le cours du Professeur Nievergelt :
`http://www.jn.inf.ethz.ch/education/script/chapter5.pdf`.

Exercice 5.8 *Soit la grammaire $S \rightarrow SS|a$.*

1. *Montrez qu'elle engendre a^+ l'ensemble des suites non vides de a.*
2. *Montrez que cette grammaire est ambigüe*
3. *Soit X_n le nombre de dérivations gauches de a^n. Prouvez que $X_{n+1} = (1/(n+1))\binom{2n}{n}$*

Chapitre IV

Automates finis

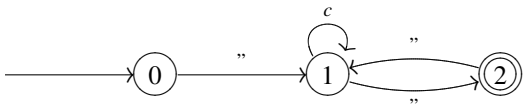
1 Introduction

Un automate fini est une machine abstraite qui effectue des calculs en utilisant une mémoire de taille bornée. Il n'est pas possible d'augmenter la taille de la mémoire au delà de cette borne. Les tailles des données et des résultats peuvent être supérieurs à cette borne puisque les données sont lues, les résultats produits progressivement au cours du calcul. La longueur d'un calcul peut aussi être supérieure à la taille de la mémoire de l'automate ; celui-ci passe alors nécessairement plusieurs fois par un même état de sa mémoire. Les automates finis ont des applications importantes : définition de certains aspects des langages naturels ou artificiels (en particulier de la lexicographie), description de machines physiques (circuits électroniques, machines à calculer, distributeur d'objets, etc.), définition de protocoles de communication dans des réseaux, description de systèmes de commandes (comme le système de commandes d'un ascenseur), etc. Les automates finis peuvent être utilisés pour calculer des fonctions, ou pour reconnaître des langages. On montre que les langages reconnus par les automates finis coïncident avec les langages réguliers, c'est-à-dire les langages qui sont les valeurs des expressions régulières. Les langages réguliers comprennent, outre les langages finis, les langages infinis que l'on peut définir en utilisant l'opération de concaténation itérée (l'étoile). Un ordinateur - en faisant abstraction des mémoires périphériques - peut être décrit par un automate fini, puisque sa mémoire centrale est de taille bornée. Mais ce type de descriptions n'a que peu d'intérêt, au vu de la taille très grande des mémoires des ordinateurs. Un meilleur modèle est fourni par les machines de Turing, qui utilisent une mémoire finie non bornée.

Exemple 1.1 (Automate fini reconnaissant les chaînes de caractères d'un langage de programmation) *Cet ensemble est défini informellement par :*
toute chaîne commence et se termine par le caractère " (guillemets) ; toute occurrence du caractère " (guillemets) à l'intérieur d'une chaîne doit être doublée. Par exemple, la chaine de caractères je dis "oui" s'écrit : "je dis ""oui""" ; la chaîne de caractères de longueur 1 se composant d'un seul guillemet s'écrit : """" .

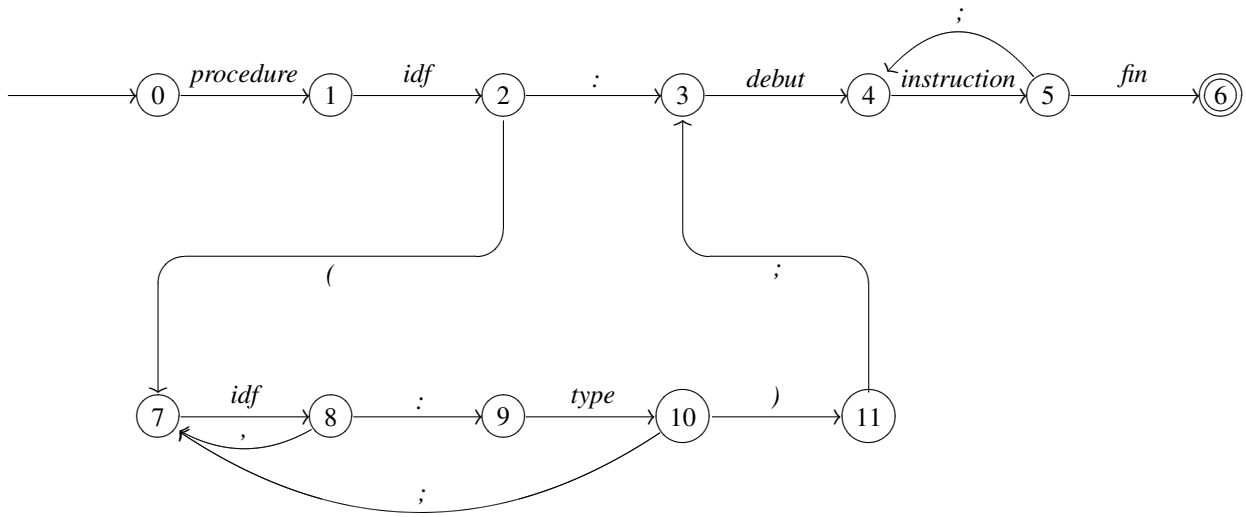
L'ensemble peut être défini formellement par l'expression régulière : "(c + "")*"*
où les opérateurs des expressions régulières sont notés en gras, et c désigne un caractère quelconque différent de " .

Le dessin ci-dessous représente un automate fini reconnaissant l'ensemble des chaînes de caractères : les chaînes de caractères sont les chemins reliant le sommet le plus à gauche (l'état initial de l'automate) au sommet le plus à droite (l'état final de l'automate). Comme précédemment, le symbole c désigne un caractère quelconque différent du caractère ". L'automate a trois états :



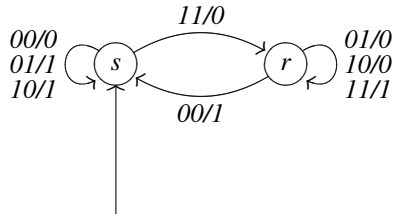
Exemple 1.2 (Automate fini décrivant les déclarations de procédure d'un langage de programmation.) *Informellement , une déclaration de procédure commence par le mot clé procédure, suivi d'un identificateur qui est le nom de la procédure. On a ensuite la déclaration de la liste des paramètres (cette partie peut être vide), puis, après un " ;", le corps de la procédure. Chaque paramètre est un identificateur, auquel est associé un type. On sépare un paramètre de*

son type par le caractère " :". On peut mettre en facteur des paramètres ayant la même type. Le corps de la procédure commence par le mot clé début, suivi d'une liste non vide d'instructions, et se termine par le mot clé fin. Les instructions sont séparées par le caractère " ;".
Dans l'automate ci-dessous, qui contient 12 états, les déclarations de procédures sont définies par l'ensemble des chemins de l'état initial (en haut à gauche) à l'état final (en haut à droite). On n'a pas défini les identificateurs, les types et les instructions. Il n'y a pas de difficulté à définir les identificateurs avec un automate fini. Par contre, si les instructions peuvent contenir des déclarations de procédures, on ne peut pas décrire les déclarations de procédure par un automate fini : un formalisme équivalent aux grammaires hors-contexte est alors nécessaire.



Nous verrons au paragraphe 7 de ce chapitre que les fonctionnalités d'un automate fini peuvent être étendues en lui ajoutant une fonction de sortie. L'exemple ci-dessous montre comment se servir d'un tel automate pour calculer la somme de deux entiers binaires.

Exemple 1.3 (Automate fini calculant la somme de deux entiers notés en binaire) *Les deux entiers à additionner sont écrits sur le même nombre de chiffres binaires. L'automate lit simultanément les deux opérandes, en commençant pas les chiffres des unités, et produit le résultat chiffre par chiffre en commençant par le chiffre des unités. Ses entrées sont donc les couples de chiffres binaires. Pour chaque entrée, l'automate produit un résultat qui est un chiffre binaire. La notation 00 / 1 signifie : pour l'entrée 00, on produit 1. L'automate a deux états : l'état s correspond à l'absence de retenue, l'état r à la présence d'une retenue. Deux entiers à additionner définissent un chemin de l'automate, commençant par l'état sans retenue s. Si ce chemin se termine en s, il définit aussi le résultat de l'addition ; s'il se termine en r, il faut encore écrire un 1 pour la retenue.*



La suite de ce chapitre comprend :
- La définition des automates finis reconnaissant des langages, comme ceux des exemples 1.1 et 1.2. On consultera le paragraphe 2.
- Les preuves de l'équivalence des automates finis avec des automates finis particuliers : automates sans ε transitions (paragraphe 4), automates déterministes (paragraphe 5).
- La preuve de l'équivalence des automates finis et des expressions régulières fait l'objet du chapitre V.
- Le problème de la minimisation des automates finis est traité au chapitre VI.

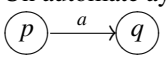
– Les automates définissant des fonctions, comme celui de l'exemple 1.3, - encore appelés machines finies - sont étudiés brièvement au paragraphe 7 du présent chapitre.

2 Automates finis et langages

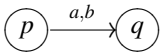
Définition 2.1 (automate fini) *Un* automate fini *est un quintuplet* $< Q, V, \delta, I, F >$ *où :*
- *Q est un ensemble fini d'états*
- *V est un ensemble fini de symboles (le vocabulaire de l'automate)*
- *δ est un sous-ensemble de $Q \times V \cup \{\varepsilon\} \times Q$ appelé l'ensemble des transitions*
- *I est un sous-ensemble de Q, l'ensemble des états initiaux*
- *F est un sous-ensemble de Q, l'ensemble des états finals*

Une transition de δ telle que (p,a,q) est appelée une a-transition. L'état p est l'origine et l'état q est l'extrémité de cette transition.
On peut aussi voir δ comme une application de $Q \times V \cup \{\varepsilon\}$ dans $\mathcal{P}(Q)$ en posant pour $p \in Q$, $a \in V \cup \{\varepsilon\}$:
$\delta(p,a) = \{q \mid (p,a,q) \in \delta\}$

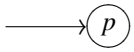
Un automate ayant δ comme ensemble de transitions est dessiné avec les conventions suivantes :



signifie que $(p,a,q) \in \delta$



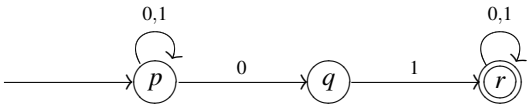
signifie que $(p,a,q) \in \delta$ et $(p,b,q) \in \delta$



indique que p est un état initial

indique que p est un état final

Exemple 2.1 *D'après ces conventions, l'automate ci-dessous :*



a comme état initial p, comme état final r et comme ensemble de transitions $\{(p,0,p),(p,1,p),(p,0,q),(q,1,r),(r,0,r),(r,1,r)\}$.

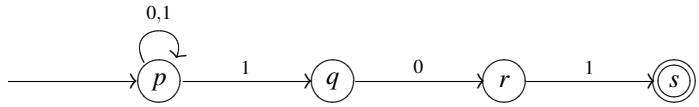
Définition 2.2 (chemins et traces) *Soit $A =< Q, V, \delta, I, F >$ un automate fini.*
Un chemin *de A de longueur n est une suite de transitions (r_i, a_{i+1}, r_{i+1}) où $0 \leq i < n$. Ce chemin mène de l'état r_0 à l'état r_n avec la* trace *$a_1 \ldots a_n$. Par convention, la suite vide est un chemin de longueur 0 et de trace ε conduisant d'un état à lui-même.*

Dans l'exemple ci-dessus, $(p,1,p)(p,0,q)(q,1,r)(r,1,r)$ est un chemin de p à r dont la trace est 1011.

Définition 2.3 (mots et langage reconnu par un automate) *Le mot x est* reconnu *(ou* accepté*) par un automate s'il y a un chemin de cet automate de trace x menant d'un état initial à un état final.*
On note $L(A)$ l'ensemble des mots reconnus par l'automate A. $L(A)$ est le langage reconnu *par l'automate A.*
Un langage reconnu par un automate fini est appelé langage d'états finis.
Deux automates sont équivalents s'ils reconnaissent le même langage.

Dans l'exemple ci-dessus, le mot 01 est reconnu et le langage reconnu par l'automate est défini par l'expression régulière $(0+1)^*01(0+1)^*$, qui caractérise l'ensemble de mots sur $\{0,1\}$ comportant le sous-mot 01.

Exemple 2.2 *Considérons l'automate suivant sur le vocabulaire* $\{0,1\}$ *:*



$(p,0,p)(p,1,q)(q,0,r)(r,1,s)$ *est un chemin menant de* p *à* s *avec la trace* 0101. *Ce mot est reconnu par l'automate car* p *est un état initial et* s *est un état final.*
Le langage reconnu par l'automate est défini par l'expression régulière $(0+1)^*101$, *qui caractérise l'ensemble de mots sur* $\{0,1\}$ *terminés par* 101.

Exercice 2.1 *Définir un algorithme, qui, étant donné un automate fini* A *et une chaîne* x *sur le vocabulaire de* A, *décide si* $x \in L(A)$.

3 Décomposition des traces

Propriété 3.1 *Soient* A *un automate fini de vocabulaire* V, x *et* y *deux chaînes sur* V, p *et* q *deux états de* A.
Un chemin mène de p *à* q *avec la trace* xy *si et seulement si il existe un état* r *tel qu'un chemin mène de* p *à* r *avec la trace* x *et de* r *à* q *avec la trace* y.

Preuve : La condition suffisante est évidente et nous prouvons seulement la condition nécessaire. Soit un chemin (r_i, a_{i+1}, r_{i+1}) pour i de 0 à $n-1$ menant de l'état p à l'état q avec la trace xy. Puisque les symboles de chaque transition sont éléments de V ou sont égaux à ε, il existe k tel que $0 \leq k \leq n$ et $x = a_1 \ldots a_k$. Par suite, il y a un chemin conduisant de p à r_k avec la trace x et de r_k à q avec la trace y. □

Propriété 3.2 *Soient* A *un automate fini de vocabulaire* V, a *un élément de* V *et* p,q *deux états de* A. *Il y a un chemin de trace* a *entre* p *et* q, *si et seulement si il existe deux états* r *et* s *tels que, un chemin de trace* ε *mène de* p *à* r, (r,a,s) *est une transition et un chemin de trace* ε *mène de* s *à* q.

On laisse au lecteur le soin de rédiger cette preuve.

4 Automates finis sans ε-transition

Théorème 4.1 *Tout automate fini est équivalent à un automate fini sans* ε *transition*

Preuve : Soit $A =< Q,V,\delta,I,F >$ un automate fini.
Soit $B =< Q,V,\eta,I,G >$ l'automate fini sans ε transition ainsi défini :
- Pour tout $a \in V$, pour tous $p,q \in Q$,
  $(p,a,q) \in \eta$ si et seulement si il existe un état r avec un chemin de A de trace ε conduisant de p à r et $(r,a,q) \in \delta$.
- $G = F \cup \{p \in Q \mid$ il y a un chemin de trace ε entre p et un état de $F\}$

On peut décrire autrement la construction de B :
- Pour tout $p \in Q$, soit $f(p) = \{q \in Q \mid$ il y a un chemin de trace ε entre p et $q\}$
- Pour tout $a \in V$, pour tout $p \in Q$, $\eta(p,a) = \bigcup_{r \in f(p)} \delta(r,a)$
- $G = F \cup \{p \in Q \mid f(p) \cap F \neq \emptyset\}$

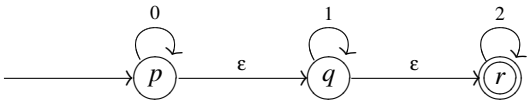
Prouvons que les automates A et B sont équivalents.
On montre tout d'abord la propriété $P(x)$ suivante :

> Il existe un chemin de A de trace x entre l'état p et l'état q si et seulement si il y a un état r et un chemin de B menant de p à r avec la trace x et un chemin de A menant de r à q avec la trace ε.

On procède par induction sur la longueur de x.
Pour $x = \varepsilon$, $P(x)$ est vérifiée en choisissant $r = p$. Supposons $P(x)$ vraie et montrons $P(ax)$ pour $a \in V$.
Il a un chemin de A de trace ax entre p et q
si et seulement si (d'après la Propriété 3.1) il existe un état t tel qu'un chemin de A mène de p à t avec la trace a, et un chemin de A mène de t à q avec la trace x

si et seulement si (d'après la Propriété 3.2) il y a des états r,s,t tels qu'un chemin de A de trace ε mène de p à r, $(r,a,s) \in \delta$, un chemin de A de trace ε mène de s à t, et un chemin de A de trace x mène de t à q
si et seulement si (par définition de B) il y a des états s,t tels que $(p,a,s) \in \eta$, un chemin de A de trace ε mène de s à t, et un chemin de A de trace x mène de t à q
si et seulement si (par définition des traces) il y a des états s,t tels que $(p,a,s) \in \eta$, un chemin de A de trace x mène de s à q
si et seulement si (d'après l'hypothèse d'induction) il y a des états s,u tels que $(p,a,s) \in \eta$, un chemin de B de trace x mène de s à u et un chemin de A de trace ε mène de u à q
si et seulement si (par définition des traces) il y a un état u tels que un chemin de B de trace xa mène de p à u et un chemin de A de trace ε mène de u à q
Ce qui établit la propriété $P(ax)$, donc prouve $P(x)$ pour toute chaîne x.
On en déduit l'équivalence des deux automates. Soit $x \in V^*$:
$x \in L(A)$
si et seulement si il y a un chemin de A de trace x entre un état initial et un état final de A
si et seulement si (d'après la propriété $P(x)$) il existe un état r et un chemin de B de trace x entre un état initial de B et r et un chemin de A de trace ε entre r et un état final de A
si et seulement si (par définition des états finals de B) il y a un chemin de B de trace x entre un état initial et un état final de B
si et seulement si $x \in L(B)$ □

Exemple 4.1 *On élimine les ε transitions de l'automate A ci-dessous, ayant pour vocabulaire $\{0,1,2\}$:*



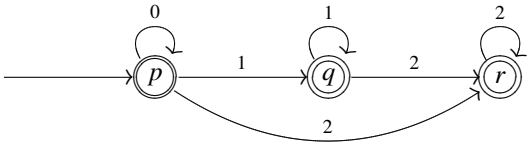
Pour chaque état s, on calcule $f(s)$ l'ensemble des états atteignables avec la trace ε :

s	$f(s)$
p	$\{p,q,r\}$
q	$\{q,r\}$
r	$\{r\}$

Puisqu'il y a des chemins de trace ε entre p et l'état final r ainsi qu'entre q et r, ces deux états deviennent aussi des états finals.
La fonction de transition de l'automate sans ε est présentée sous forme d'une table dont les cases sont les valeurs de $\eta(s,a)$ pour chaque état s et symbole a.

	0	1	2
p	$\{p\}$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{q\}$	$\{r\}$
r	$\emptyset$	$\emptyset$	$\{r\}$

Dessinons ce même automate :



5 Automates finis déterministes

Définition 5.1 (automate déterministe) *Un automate fini* déterministe *est un automate fini $< Q,V,\delta,I,F >$ n'ayant qu'un état initial, sans ε-transition, tel que pour tout état p et tout symbole a (un symbole est un élément de V), il y a* un et un seul état *q tel que $(p,a,q) \in \delta$.*

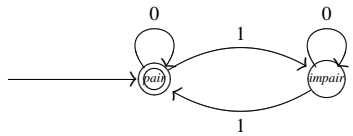
Dans la description d'un tel automate, l'ensemble I des états initials est remplacé par son seul état. Soit p un état et a un symbole, puisqu'il n'y a qu'un seul état q tel que $(p,a,q) \in \delta$, on note cet état par $\delta(p,a)$ et on voit δ comme

une application de $Q \times V$ dans Q. Avec ce formalisme, la fonction δ est donc *totale*. Il est possible de trouver dans la littérature des formalismes où δ est une fonction partielle, c'est-à-dire que pour tout $p \in Q$ et $a \in V$, il existe *au plus* un état tel que $(p,a,q) \in \delta$.

Définition 5.2 (extension de la fonction de transition) *Soit $\delta : Q \times V \mapsto Q$ la fonction de transition d'un automate déterministe.*
Soit $\delta^* : Q \times V^* \mapsto Q$ la fonction ainsi définie :
- $\delta^*(p,\varepsilon) = p$ *pour tout* $p \in Q$
- $\delta^*(p,ax) = \delta^*(\delta(p,a),x)$ *pour tout* $p \in Q$, $a \in V$, $x \in V^*$

Remarquons que pour p un état et a un symbole, $\delta^*(p,a) = \delta^*(\delta(p,a),\varepsilon) = \delta(p,a)$. Donc la fonction δ^* est une extension de δ et dans la suite nous ne distinguerons plus ces deux fonctions, et noterons δ à la place de δ^*.

Exemple 5.1 *L'automate A dessiné ci-dessous est un automate déterministe sur le vocabulaire $\{0,1\}$: de chaque état part une et une seule 0-transition ainsi que une et une seule 1 transition.*



$L(A)$ est l'ensemble de chaînes ayant un nombre pair de 1. Notez le choix du nom des états : toute trace d'un chemin de l'état pair à l'état pair comprend un nombre pair de 1, toute trace d'un chemin de l'état pair à l'état impair comprend un nombre impair de 1.

Propriété 5.1 *Soit $\delta : Q \times V \mapsto Q$ la fonction de transition d'un automate déterministe. Soient p,q deux étatx et $x \in V^*$. $\delta(p,x) = q$ si et seulement si il y a un chemin de trace x menant de p à q.*

La preuve est immédiate par induction sur la longueur de x.

Propriété 5.2 *Soit $A =< Q,V,\delta,i,F >$ un automate déterministe. $L(A) = \{x \in V^* \mid \delta(i,x) \in F\}$*

Preuve : Soit $x \in V^*$. D'après la propriété 5.1 :
$\delta(i,x) \in F$ si et seulement si il y a un chemin de trace x entre l'état initial et un état final de A.
D'après la définition 2.3, cela veut dire que $x \in L(A)$. □

Propriété 5.3 *Soit $\delta : Q \times V \mapsto Q$ la fonction de transition d'un automate déterministe. Pour tout état $p \in Q$ et toutes chaines $x,y \in V^*$, on a : $\delta(p,xy) = \delta(\delta(p,x),y)$.*

Preuve : On procède par induction sur la longueur de x.
Pour $x = \varepsilon$, d'après la définition 5.2, $\delta(\delta(p,x),y) = \delta(p,y)$. Puisque $y = xy$, on a bien : $\delta(p,xy) = \delta(\delta(p,x),y)$.
Supposons l'égalité vérifiée pour x et montrons qu'elle reste vérifiée pour la chaîne ax où $a \in V$:

$\delta(p,axy) = \delta(\delta(p,a),xy)$	Par définition 5.2
$\delta(\delta(p,a),xy) = \delta(\delta(\delta(p,a),x),y)$	Par hypothèse d'induction
$\delta(\delta(\delta(p,a),x),y) = \delta(\delta(p,ax),y)$	Par définition 5.2

Donc $\delta(p,axy) = \delta(\delta(p,ax),y)$ □

Définition 5.3 (État accessible, automate connecté) *Soit $A =< Q,V,\delta,i,F >$ un automate fini déterministe. L'état p est* accessible *si et seulement si il y une chaîne $x \in V^*$ telle que $p = \delta(i,x)$. Un automate dont tous les états sont accessibles est dit* connecté.

Exercice 5.1 *Écrire un algorithme calculant les états accessibles d'un automate fini déterministe.*

Théorème 5.1 (Élimination des états inaccessibles) *Tout automate fini déterministe est équivalent à un automate fini déterministe connecté.*

Preuve : Soit $A =< Q,V,\delta,i,F >$ un automate fini déterministe et R l'ensemble de ses états accessibles.
L'automate B , obtenu en supprimant les états inaccessibles, où $B =< R,V,\eta,i,R \cap F >$ où $\eta = \delta \cap (R \times V \times R)$, est un automate fini déterministe connecté équivalent à A.

1. L'automate B est déterministe.
   Puisque l'automate A est déterministe, pour tout état $p \in R$, tout symbole $a \in V$, il y a *au plus* un état $q \in R$ tel que $(p,a,q) \in \eta$.
   Montrons qu'il y a au moins un tel état. Soit $p \in R$ et $a \in V$. Puisque p est accessible, il existe une chaine x telle que $\delta(i,x) = p$ et par la propriété 5.3 de l'extension de δ, on a $\delta(p,a) = \delta(\delta(i,x),a) = \delta(i,xa)$. Donc l'état $\delta(p,a)$ est aussi accessible et $(p,a,\delta(p,a)) \in \eta$.
   Par suite η est une fonction de $R \times V$ dans R.
2. L'automate B est connecté.
   D'après la preuve ci-dessus, δ et η sont identiques sur le domaine $R \times V$, donc par une induction triviale, elles sont aussi indentiques sur le domaine $R \times V^*$.
   Soit $p \in R$, puisque p est accessible dans A, il existe une chaîne x telle $\delta(i,x) = p$. Puisque δ et η sont identiques sur le domaine $R \times V^*$, $\eta(i,x) = p$, donc p est accessible dans B.
   Donc tous les états de B sont accessibles : B est connecté.
3. Les automates A et B sont équivalents.

$x \in L(A)$	
si et seulement si $\delta(i,x) \in F$	D'après la propriete 5.2
si et seulement si $\delta(i,x) \in R \cap F$	Car l'état $\delta(i,x)$ est accessible, donc élément de R
si et seulement si $\eta(i,x) \in R \cap F$	Car δ et η sont identiques sur le domaine $R \times V^*$
si et seulement si $x \in L(B)$	D'après la propriete 5.2

□

Exercice 5.2 *Écrire un algorithme calculant un automate détermiste* connecté *équivalent à un automate déterministe donné.*

6 Construction des automates déterministes

Théorème 6.1 *Tout automate fini sans ε transition est équivalent à un automate déterministe.*

Preuve : Soit $A =< Q,V,\delta,I,F >$ un automate fini sans ε-transitions.
Soit $B =< \mathcal{P}(Q),V,\eta,I,G >$ l'automate fini déterministe, ayant I comme état initial et défini par :
$\eta(P,a) = \bigcup_{p \in P} \delta(p,a)$ pour $P \subset Q, a \in V \quad G = \{P \mid P \subset Q \text{ et } P \cap F \neq \emptyset\}$
Prouvons que ces deux automates sont équivalents.
Soit $f : \mathcal{P}(Q) \times V^* \mapsto: \mathcal{P}(Q)$ ainsi défini :
$f(P,x)$ est l'ensemble des états $q \in Q$ tels qu'il y a un chemin de trace x entre un élément de P et q.
Nous montrons d'abord que les fonctions f et η sont identiques.

1. Pour tout $P \subset Q, a \in V, x \in V^*$, on a $f(P,ax) = f(\eta(P,a),x)$. En effet

$q \in f(P,ax)$	
si et seulement si il existe $p \in P, r \in Q$, un chemin de A menant de p à r de trace a et un chemin de A menant de r à q de trace x	Par définition de f
si et seulement si il existe $p \in P, r \in Q$, $r \in \delta(p,a)$ et un chemin de A menant de r à q de trace x	Parce que A n'a pas d'ε-transition
si et seulement si il existe $r \in \eta(P,a)$ et un chemin de A menant de r à q de trace x	Par définition de η
si et seulement si $q \in f(\eta(P,a),x)$	Par définition de f

2. On montre que pour tout $P \subset Q, x \in V^*$, on a $f(P,x) = \eta(P,x)$ par induction sur x.
   Pour $x = \varepsilon$, l'égalité est vérifiée, car en absence d'ε-transition, $f(P,x) = P$ et par définition des extensions $\eta(P,x) = P$.
   Supposons la propriété vérifiée pour la chaîne x. Montrons qu'elle est aussi vérifiée par la chaîne ax où $a \in V$.

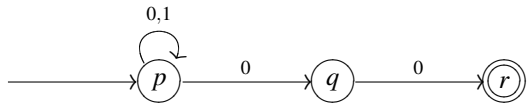
$f(P,ax) = f(\eta(P,a),x)$	D'après la propriété ci-dessus 1
$f(\eta(P,a),x) = \eta(\eta(P,a),x)$	Par induction
$\eta(\eta(P,a),x) = \eta(P,ax)$	Par la définition 5.2

3. Montrons l'équivalence des deux automates. Soit $x \in V^*$.

$x \in L(A)$	
si et seulement si il y a un chemin de trace x entre un état initial et un état final de A	Par la définition 2.3
si et seulement si $f(I,x) \cap F \neq \emptyset$	Par définition de f
si et seulement si $f(I,x) \in G$	Par définition de G
si et seulement si $\eta(I,x) \in G$	Puisque, d'après 2, les fonctions f et η sont égales
si et seulement si $x \in L(B)$	Car G est l'ensemble des états finaux de B

□

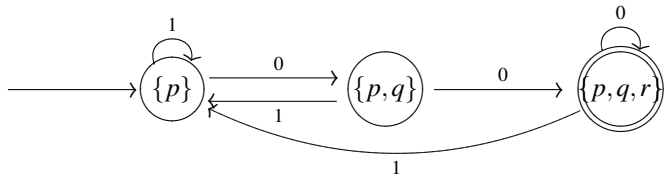
Exemple 6.1 *Soit $A =< Q,V,\delta,I,F >$ l'automate fini suivant, où $V = \{0,1\}$:*



$L(A)$ est l'ensemble des chaînes sur $\{0,1\}$ terminées par 00.
Soit $B =< \mathcal{P}(Q),V,\eta,I,G >$ l'automate déterministe construit comme il est indiqué dans le théorème 6.1 ci-dessus.
Nous représentons B par une table, puis nous dessinons l'automate obtenu en supprimant les états inaccessibles de B.

accessibles	*statut*	*états*	*0*	*1*
		$\emptyset$	$\emptyset$	$\emptyset$
(a)	*initial*	$\{p\}$	$\{p,q\}$	$\{p\}$
		$\{q\}$	$\{r\}$	$\emptyset$
	final	$\{r\}$	$\emptyset$	$\emptyset$
(b)		$\{p,q\}$	$\{p,q,r\}$	$\{p\}$
	final	$\{p,r\}$	$\{p,q\}$	$\{p\}$
	final	$\{q,r\}$	$\{r\}$	$\emptyset$
(c)	*final*	$\{p,q,r\}$	$\{p,q,r\}$	$\{p\}$

Seuls les états (a), (b), (c) sont accessibles, aussi en pratique *on construit l'automate déterministe équivalent à A en partant de son état initial $\{p\}$ et on ajoute les nouveaux états au fur et à mesure qu'ils sont accessibles donc on ajoute les lignes (b) et (c) sans construire les cinq lignes inutiles des états inacessibles.*
Après suppression des états inaccessibles (voir propriété 5.1), l'automate B est équivalent à l'automate déterministe ci-dessous, correspondant aux trois lignes (a), (b) et (c) :



Remarque 6.1 *L'automate B, dont la construction est décrite dans la preuve du théorème 6.1, comporte, en général, beaucoup d'états inaccessibles. Plutôt que d'introduire ces états puis de les éliminer, on construit la fonction de transition de B en partant de son état initial et en ajoutant uniquement les états accessibles depuis cet état.*

Corollaire 6.1 *Tout automate fini est équivalent à un automate fini déterministe connecté.*

Preuve : D'après le théorème 4.1 un automate fini est équivalent à un automate fini sans ε-transition.
D'après le théorème 6.1 un automate fini sans ε-transition est équivalent à un automate déterministe.
D'après le théorème 5.1 un automate fini déterministe est équivalent à un automate fini déterministe connecté.
Ce qui termine la preuve : notez que les constructions de ces automates sont décrites dans les preuve des théorèmes indiqués. □

Exercice 6.1 *1. Montrez que le complémentaire d'un langage d'états finis est aussi d'états finis.*
Indication : utiliser le fait qu'un automate fini est équivalent à un automate déterministe.

2. *Montrez que l'intersection de deux langages d'états finis et aussi d'états finis.*
   Indication : étant donnés deux automates déterministes $A_1 =< Q_1, V, \delta_1, i_1, F_1 >$ *et* $A_2 =< Q_2, V, \delta_2, i_2, F_2 >$*, construire un automate ayant pour ensemble d'états* $Q_1 \times Q_2$ *et qui effectue* simultanément *les transitions de* A_1 *et* A_2*.*
3. *Montrez que l'union de deux langages d'états finis et aussi d'états finis.*

7 Machines de Mealy et de Moore

Les machines étudiées dans cette section sont des automates finis déterministes munis de sorties. Ces machines sont très importantes car elles modélisent le fonctionnement des processeurs, qui changent d'états en fonction des signaux d'entrée et produisent des signaux en sortie.
Avec des sorties associées aux transitions, on obtient *les machines de Mealy*. Lorsque les sorties sont associées aux états, on obtient *les machines de Moore*.
Les automates finis déterministes peuvent être vus comment des cas particuliers de ces machines.

Définition 7.1 (Machine de Mealy) *Une machine Mealy est un sextuplet* $A =< Q, V, W, i, \delta, s >$ *où*
- Q *est un ensemble fini d'états*
- V *est le vocabulaire d'entrée*
- W *est le vocabulaire de sortie*
- $i \in Q$ *est l'état initial*
- $\delta : Q \times V \mapsto Q$ *est la fonction de transition*
- $s : Q \times V \mapsto W$ *est la fonction de sortie*

La fonction δ *est étendue en une fonction de* $Q \times V^*$ *dans* Q *comme dans le cas des automates déterministes, on rappelle que :*

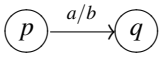
$\delta(p, \varepsilon) = p$ *pour* $p \in Q$
$\delta(p, ax) = \delta(\delta(p, a), x)$ *pour* $p \in Q, a \in V, x \in V^*$

La fonction s *est étendue en une fonction de* $Q \times V^*$ *dans* W^* *avec :*

$s(p, \varepsilon) = \varepsilon$ *pour* $p \in Q$
$s(p, ax) = s(p, a)s(\delta(p, a), x)$ *pour* $p \in Q, a \in V, x \in V^*$

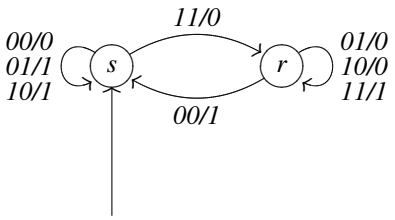
Soit $s_A : V^* \mapsto W^*$ *la fonction définie par* $s_A(x) = s(i, x)$*.*
Cette fonction est la fonction (d'entrée-sortie) calculée par A*.*
Deux machines de Mealy (ayant même vocabulaire d'entrée et même vocabulaire de sortie) sont équivalentes si elles calculent la même fonction d'entrée-sortie.

L'état initial d'une telle machine est noté comme celle d'un automate fini. La fonction de transition et la fonction d'entrée-sortie sont notées avec les conventions suivantes :



signifie que $\delta(p, a) = q$ et $s(p, a) = b$

Exemple 7.1 *On trouve un premier exemple de machine de Mealy en 1.3 avec un additionneur binaire. Nous le redonnons ici :*



La machine a deux états s *(pas de retenue) et* r *(retenue). Son vocabulaire d'entrée est* $V = \{00, 01, 10, 11\}$*, son vocabulaire de sortie est* $W = \{0, 1\}$*.*
Pour ajouter 110 *(6 en décimal) et* 110*, on donne le mot d'entrée* 00.11.11 *dans lequel les unités sont mises à* gauche *et le point est utilisé pour séparer les symboles d'entrée.*

Appelons A cette machine, avec les même notations que dans la définition ci-dessus. On a :
$s_A(00.11.11) = 001$ *et* $\delta(s, 00.11.11) = r$. *Autrement dit la somme vaut* 1100 *(*12 *en décimal), le* 1 *en tête venant de la retenue et* 100 *venant de la sortie retournée.*

Propriété 7.1 *Soit* $< Q, V, W, i, \delta, s >$ *une machine de Mealy. Pour tous* $p \in Q, x, y \in V^*$, *on a :*
$s(p, xy) = s(p,x)s(\delta(p,x), y)$

Exercice 7.1 *Prouver la propriété ci-dessus.*

Définition 7.2 (Machine de Moore) *Une machine Moore est un sextuplet* $A =< Q, V, W, i, \delta, s >$ *où*
- Q *est un ensemble fini d'états*
- V *est le vocabulaire d'entrée*
- W *est le vocabulaire de sortie*
- $i \in Q$ *est l'état initial*
- $\delta : Q \times V \mapsto Q$ *est la fonction de transition*
- $s : Q \mapsto W$ *est la fonction de sortie*

La fonction s est étendue en une $Q \times V^*$ *dans* W^* *avec :*

$s(p, \varepsilon) = s(p)$ *pour* $p \in Q$
$s(p, ax) = s(p)s(\delta(p,a), x)$ *pour* $p \in Q, a \in V, x \in V^*$

La fonction d'entrée-sortie calculée par A est définie comme pour la machine de Mealy, en démarrant la machine à partir de son état initial. L'équivalence entre deux machines de Moore est définie comme pour les machines de Mealy.

Exercice 7.2 *Soit* $A =< Q, V, i, \delta, F >$ *un automate fini déterministe. Définir une machine de Moore* $B =< Q, V, \{0,1\}, i, \delta, s >$ *telle que pour toute chaîne* $x \in V^*$, *x est reconnu par A si et seulement si* $s_B(x)$ *est une chaîne qui se termine par* 1.

Cet exercice montre que les automates finis déterministes peuvent être vues comme des machines de Moore particulières.
Nous montrons aussi comment passer d'une machine de Moore à une machine de Mealy et inversement.

Théorème 7.1 (De Moore à Mealy) *Soit* $A =< Q, V, W, i, \delta, s >$ *une machine de Moore.*
Soit $B =< Q, V, W, i, \delta, t >$ *la machine de Mealy telle que : pour tout* $p \in Q, a \in V, t(p,a) = s(\delta(p,a))$.
Pour tout $p \in Q, x \in V^*$, *nous avons :* $s(p,x) = s(p)t(p,x)$.

La preuve est laissée en exercice.

Théorème 7.2 (De Mealy à Moore) *Soit* $A =< Q, V, W, i, \delta, s >$ *une machine de Mealy. La transformation en une machine de Moore est résumé par le schéma suivant :*

Machine de Mealy *Machine de Moore*

Autrement dit, le symbole b, que la machine de Mealy doit écrire en lisant a dans l'état p, est mémorisé dans l'état (q,b) *de la machine de Moore qui écrit b dans cet état. Formalisons cette construction.*
Soit $B =< Q \times V, V, W, (i,a), \eta, t >$ *la machine de Moore suivante :*
- *l'état initial est* (i,a) *où* $a \in V$ *est un symbole quelconque*
- $\eta((p,a),b) = (\delta(p,b), s(p,b))$ *pour* $p \in Q$, $a, b \in V$ *: la transition de la machine de Mealy est effectuée et le symbole qu'elle devait écrire lors de cette transition est mémorisée.*
- $t((p,a)) = a$ *pour* $p \in Q, a \in A$

Pour tout $p \in Q, a \in V, x \in V^*$, *nous avons :*$t((p,a),x) = as(p,x)$

La preuve est laissée en exercice. On note que cette transformation multiplie le nombre d'états de la machine de Mealy par le nombre de symboles du vocabulaire d'entrée.

Chapitre V

Equivalence des automates finis et des expressions régulières

Ce chapitre est consacré à la démonstration du théorème : un langage est reconnu par un automate fini si et seulement si il est la valeur d'une expression régulière. La démonstration de ce résultat est constructive : elle définit des algorithmes de traduction des expressions régulières en automates finis équivalents et, pour la réciproque, de traduction des automates finis en expressions régulières.

1 Tout langage régulier est reconnu par un automate fini

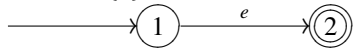
Dans ce paragraphe sauf exception, on identifie une expression régulière et le langage qui lui est associé.

Définition 1.1 (Automate simple) *Un automate fini est simple s'il n'a qu'un état initial et qu'un état final, si aucune transition n'a pour extrémité cet état initial et si aucune transition n'a pour origine cet état final.*

Théorème 1.1 *Tout langage régulier est reconnu par un automate fini.*

Preuve : On prouve la propriété suivante :
(1) Soit e une expression régulière sur V. Il existe un entier α (où $\alpha \geq 2$) et un automate simple $A =< \{1,2,\ldots,\alpha\}, V, \delta, \{1\}, \alpha >$ qui reconnait e.
Nous montrons cette propriété par récurrence "structurelle" sur e.
Si $e = \emptyset$, l'expression est reconnue par l'automate simple :

Si $e \in V \cup \{\varepsilon\}$, l'expression est reconnue par l'automate simple :

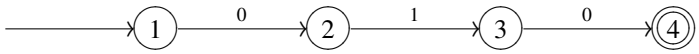


Supposons que la propriété soit vérifiée par les expressions f et g. D'après cette hypothèse de récurrence, il existe deux entiers α et β où $\alpha \geq 2$ et $\beta \geq 2$, un automate simple $A =< \{1,\ldots,\alpha\}, V, \delta, \{1\}, \{\alpha\} >$ reconnaissant f et un automate simple $B =< \{1,\ldots,\beta\}, V, \eta, \{1\}, \{\beta\} >$ reconnaissant g. et montrons qu'elle est vérifiée par e ayant l'une des formes $f^*, f+g, fg$.

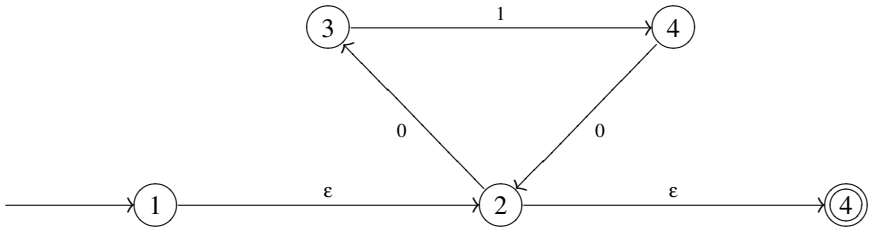
1. cas $e = f^*$
   Soit l'automate $C =< \{1,\ldots,\alpha+1\}, V, \zeta, \{1\}, \{\alpha+1\} >$ où C est obtenu, en «collant» l'état initial et final de A, c'est-à dire en changeant les états $1,\ldots,\alpha-1,\alpha$ de l'automate A en les états $2,\ldots,\alpha,2$, et en ajoutant les deux transitions $(1,\varepsilon,2)$ et $(2,\varepsilon,\alpha+1)$.
   Plus précisément
   $$\begin{aligned}\zeta &= \{(p+1,a,q+1) \mid (p,a,q) \in \delta \text{ et } q < \alpha\} \\ &\cup \{(p+1,a,2) \mid (p,a,\alpha) \in \delta\} \\ &\cup \{(1,\varepsilon,2),(2,\varepsilon,\alpha+1)\}\end{aligned}$$

Il est évident, en examinant ζ, que l'automate C est simple. Nous montrons que C reconnait e.

Exemple 1.1 *Le langage* 010 *est reconnu par l'automate simple*



Avec la construction indiquée, le langage $(010)^*$ *est reconnu par l'automate simple*



On montre que $L(C) = L(A)^*$.

(a) $L(A)^* \subset L(C)$.
On montre que C vérifie :
(a) Si $x \in L(A)$ alors il y a un chemin de C de trace x de l'état 2 à l'état 2.
En effet, soit $x \in L(A)$. Il existe $n \in \mathbb{N}$ et une suite de transitions, $(r_i, a_{i+1}, r_{i+1}) \in \delta$ pour i de 0 à $n-1$ telle que : $r_0 = 1, r_n = \alpha, x = a_1 \dots a_n$.
Puisque l'automate A est simple, l'état α n'est l'origine d'aucune transition, donc pour i de 0 à $n-1$, $1 \leq r_i < \alpha$.
Par définition de ζ, il en résulte que pour i de 0 à $n-2$, $(r_i, a_{i+1}, r_{i+1}) \in \zeta$, et que $(r_{n-1}, a_n, 2) \in \zeta$.
Donc il y a un chemin de C de trace x de l'état 2 à l'état 2 : la propriété (a) est établie.
Soit $x \in L(A)^*$.
Il existe $n \in \mathbb{N}$ tel que $x = y_1 \dots y_n$ et pour i de 1 à n,$y_i \in L(A)$.
D'après la propriété (a), pour i de 1 à n, il y a un chemin de C de trace y_i de l'état 2 à l'état 2, donc il existe un chemin s de C de trace x de l'état 2 à l'état 2. Par suite le mot x est reconnu par C grâce au chemin $(1, \varepsilon, 2)s(2, \varepsilon, \alpha+1)$. Donc $x \in L(C)$.

(b) $L(C) \subset L(A)^*$.
On montre par récurrence sur n que l'automate C vérifie ;
(b) S'il y a un chemin de C de trace x et de longueur n entre l'état 2 et l'état 2, alors $x \in L(A)^*$.
Supposons cette propriété vérifiée pour tout chemin de longueur inférieure à n.
Soit un chemin de C de trace x et de longueur n entre l'état 2 et l'état 2.
Il existe une suite de transitions $(r_i, a_{i+1}, r_{i+1}) \in \zeta$ pour i de 0 à $n-1$ telle que $r_0 = 2, r_n = 2, x = a_1 \dots a_n$.
Si $n = 0$ alors $x = \varepsilon$ donc $x \in L(A)^*$, ce qui prouve (b).
Si $n > 0$ alors il y a un plus petit entier j tel que $j > 0$ et $r_j = 2$.
Puisque l'automate C est simple, ni l'état 1, ni l'état $\alpha+1$ ne figurent sur un chemin de l'état 2 à l'état 2.
Donc par définition de ζ, pour i de 0 à $j-2$, $(r_i - 1, a_{i+1}, r_{i+1} - 1) \in \delta$, et $(r_{j-1}, a_j, \alpha) \in \delta$.
Ainsi il y a un chemin de A de trace $a_1 \dots a_j$ de l'état 1 à l'état α, autrement dit $a_1 \dots a_j \in L(A)$.
Par définition j, il y a un chemin de C de longueur $n-j$, donc inférieure à n, entre l'état 2 et l'état 2 : par hypothèse de récurrence, $a_{j+1}, \dots, a_n \in L(A)^*$.
Donc $x = a_1 \dots a_n \in L(A)^*$.

Ainsi $L(C) = L(A)^* = f^* = e$. Donc C reconnait e.

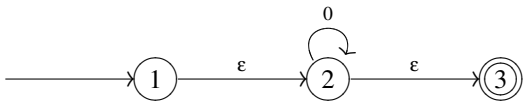
2. cas $e = f + g$.
Soit l'automate $C =< \{1, \dots, \alpha+\beta-2\}, V, \zeta, \{1\}, \{\alpha+\beta-2\} >$ où C est obtenu en «collant» d'une part l'état initial de A avec celui de B, et d'autre part l'état final de A avec celui de B.
Ce collage est effectué en changeant l'état α de l'automate A en $\alpha+\beta-2$ et en changeant pour i de 2 à β, l'état i de B en $i+\alpha-2$.

Plus précisément

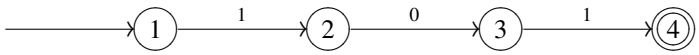
$$\begin{aligned}\zeta &= \{(p,a,q) \mid (p,a,q) \in \delta \text{ et } q < \alpha\} \cup \{(p,a,\alpha+\beta-2) \mid (p,a,\alpha) \in \delta\} \\ &\quad \cup \{(1,a,p+\alpha-2) \mid (1,a,p) \in \eta\} \\ &\quad \cup \{(p+\alpha-2,a,q+\alpha-2) \mid (p,a,q) \in \eta \text{ et } 1 < p\}\end{aligned}$$

Il est évident que C est un automate simple. Nous montrons que C reconnait e.

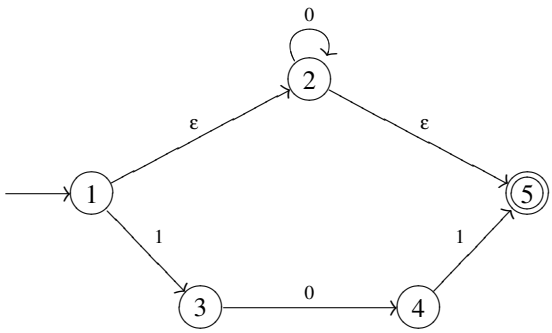
Exemple 1.2 *Le langage* 0^* *est reconnu par l'automate simple*



Le langage 101 *est reconnu par l'automate simple*



Avec la construction indiquée, le langage $0^* + 101$ *est reconnu par l'automate simple*

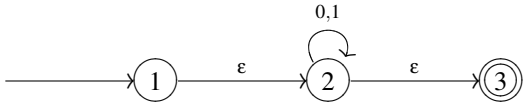


Un chemin de A de l'état 1 à l'état α devient après renommage des états un chemin de C de l'état 1 à l'état $\alpha+\beta-2$, un chemin de B de l'état 1 à l'état β devient aussi après renommage des états un chemin de C de l'état 1 à l'état $\alpha+\beta-2$.
Donc $L(A) \cup L(B) \subset L(C)$.
Réciproquement puisque les automates A et B sont simples et que l'automate C est obtenu en «collant» d'une part l'état initial de A avec celui de B, et d'autre part l'état final de A avec celui de B, tout chemin de C, entre l'état 1 et l'état $\alpha+\beta-2$, est, au nom près des états, soit un chemin de A de l'état 1 à l'état α, soit un chemin de B de l'état 1 à l'état β. Donc $L(C) = L(A) \cup L(B)$.
Puisque $L(A) = f, L(B) = g, ef+g$, C reconnait e.

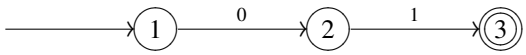
3. cas $e = fg$
Soit l'automate $C =< \{1,\ldots,\alpha+\beta-1\}, V, \zeta, \{1\}, \{\alpha+\beta-1\} >$ où C est obtenu en «collant» l'état final de A et l'état initial de B.
Pour i de 1 à β, l'état i de l'automate B devient $i+\alpha-1$.
Plus précisement

$$\zeta = \delta \cup \{(p+\alpha-1,a,q+\alpha-1) \mid (p,a,q) \in \eta\}$$

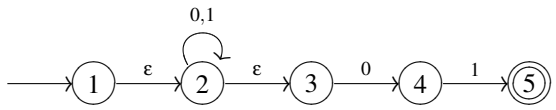
Exemple 1.3 *Le langage* $(0+1)^*$ *est reconnu par l'automate simple*



Le langage 00 *est reconnu par l'automate simple*



Avec la construction indiquée, le langage $(0+1)^*00$ *est reconnu par l'automate simple*



Il est facile de prouver que $L(C) = L(A)L(B)$. Puisque $f = L(A), g = L(B), e = fg$, l'automate C reconnait e.

□

Exercice 1.1 *Construire avec la méthode décrite dans la preuve ci-dessus, un automate fini simple reconnaissant* $((10)^*1^*)^*$.

2 Tout langage reconnu par un automate fini est un langage régulier

Dans la suite, pour dénoter les opérations sur les langages union, concaténation, on utilise les notations des expressions régulières.

Théorème 2.1 (Résolution d'une équation) *Soient* α *et* β *deux langages sur le vocabulaire* V *et* x *une variable. L'équation* $x = \alpha x + \beta$ *a une solution* $x = \alpha^*\beta$ *et cette solution est la plus petite des solutions de l'équation.*

Preuve : $x = \alpha^*\beta$ est évidemment une solution car $\alpha^*\beta = \alpha(\alpha^*\beta) + \beta$. Montrons que c'est la plus petite des solutions. Soit $x = L$ une autre solution. Par récurrence sur $i \in \mathbb{N}$, on montre que :

1. Pour tout i, $\alpha^i\beta \subset L$
   (a) $i = 0$.
   On a : $\alpha^i\beta = \{\varepsilon\}\beta = \beta$. Puisque $L = \alpha L + \beta$, on a $\alpha^i\beta \subset L$.
   (b) $i > 0$.
   Supposons $\alpha^i\beta \subset L$. Puisque la concaténation est croissante, $\alpha^{i+1}\beta \subset \alpha L$. Puique $L = \alpha L + \beta$, on a $\alpha^{i+1}\beta \subset L$. Ce qui prouve 1
2. $\alpha^*\beta \subset L$.
   Soit $u \in \alpha^*\beta$. Par définition de l'étoile, il existe $i \in \mathbb{N}$ tel que $u \in \alpha^i\beta$. Donc d'après la propriété 1, $u \in L$.

□

Remarque 2.1 *Les équations sur les langages peuvent avoir plusieurs solutions.*
Avec le vocabulaire $V = \{a\}$*, l'équation* $x = x + a$ *admet comme solutions tout langage* L *sur* V *tel que* $a \in L$*. En effet si* $a \in L$*, alors* $L = L + a$*.*
Avec le vocabulaire $V = \{a, b\}$*, l'équation* $x = (a + \varepsilon)x + b$ *a comme solutions tout langage* $a^*(L + b)$ *où* L *est un langage quelconque sur* V*. En effet* $(a + \varepsilon)a^*(L + b) + b = a^*(L + b) + b = a^*(L + b)$*.*
Avec le vocabulaire $V = \{a\}$*, l'équation* $x = ax$ *a comme unique solution* $x = \emptyset$*.*

Définition 2.1 (Système régulier d'équations) *Soit* V *un vocabulaire. Un système régulier de* $n \in \mathbb{N}$ *équations sur* V *est un ensemble de* n *variables* $x1, \ldots x_n$ *et de* n *équations où pour* i *de* 1 *à* n

$$x_i = \beta_i + \sum_{j=1}^{n} \alpha_{i,j} x_j$$

dont les coefficients $\alpha_{i,j}$ *et* β_i *sont des langages réguliers sur* V*.*

Théorème 2.2 (Résolution d'un système régulier d'équations) *Tout système régulier sur le vocabulaire* V *avec* n *variables* $x1, \ldots x_n$ *admet une plus petite solution régulière, c'est-à-dire une solution* $x_1 = L_1, \ldots, x_n = L_n$ *où* $L_1, \ldots, L_n$ *sont des langages réguliers sur* V*.*

Preuve : On prouve ce résultat par récurrence sur n et la preuve est la recherche de la solution par la méthode de Gauss : la valeur de x_n est calculée en fonction des valeurs des autres variables, puis la variable x_n est éliminée, ce qui réduit le calcul à la recherche de la solution d'un système régulier à $n-1$ variables.
Soit $S1$ le système régulier où pour i de 1 à n

$$x_i = \beta_i + \sum_{j=1}^{n} \alpha_{i,j} x_j$$

1. $n = 1$.
   La seule équation du système est $x_1 = \beta_1 + \alpha_{1,1} x_1$. D'après le théorème précédent 2.1, cette équation a comme plus petite solution $x_1 = \alpha_{1,1}^* \beta_1$.
   Puisque $\alpha_{1,1}$ et β_1 sont des langages réguliers, il en est de même de la valeur de x_1.
2. $n > 1$.
   Supposons qu'un systeme régulier à $n-1$ de variables $x_1, \ldots x_{n-1}$ a une plus petite solution régulière.
   D'après le théorème précédent 2.1, si le système $S1$ a une solution, elle vérifie

   $$x_n = \alpha_{n,n}^* (\beta_n + \sum_{j=1}^{n-1} \alpha_{n,j} x_j)$$

   Remplaçons x_n par sa valeur en fonction des variables $x_1, \ldots x_{n-1}$. On obtient le nouveau système $S2$ où pour i de 1 à $n-1$

   $$x_i = \beta'_i + \sum_{j=1}^{n-1} \alpha'_{i,j} x_j$$

   avec pour i de 1 à $n-1$

   $$\begin{aligned} \beta'_i &= \beta_i + \alpha_{i,n} (\alpha_{n,n})^* \beta_n \\ \alpha'_{i,j} &= \alpha_{i,j} + \alpha_{i,n} (\alpha_{n,n})^* \alpha_{n,j} \end{aligned}$$

   Il est clair que ce système est régulier et par définition du système $S2$, nous avons :
   $x-1 = L_1, \ldots x_{n-1} = L_{n-1}, x_n = L_n$ est la plus petite solution de $S1$
   si et seulement si
   $x-1 = L_1, \ldots x_{n-1} = L_{n-1}$ est la plus plus petite solution de $S2$
   et $L_n = \alpha_{n,n}^* (\beta_n + \sum_{j=1}^{n-1} \alpha_{n,j} L_j)$.
   Par hypothèse de récurrence, le système $S2$ a une plus petite solution régulière $x_1 = L_1, \ldots x_{n-1} = L_{n-1}$. Puisque L_n est construit par les opérations union, concaténation, étoile appliquées à des langages réguliers, L_n est un langage régulier.
   Par suite $S1$ a une plus petite solution régulière.

□

Exemple 2.1 *Cherchons la plus petite solution du système suivant sur le vocabulaire* $\{0,1\}$.

$$\begin{aligned} x_1 &= 0x_1 + 1x_2 + \varepsilon \\ x_2 &= 1x_1 + 0x_3 \\ x_3 &= 0x_2 + 1x_3 \end{aligned}$$

D'après le théorème 2.1, la plus petite solution vérifie $x_3 = 1^* 0 x_2$.
En reportant cette expression de x_3 *dans les deux premières équations, nous obtenons*

$$\begin{aligned} x_1 &= 0x_1 + 1x_2 + \varepsilon \\ x_2 &= 1x_1 + 01^* 0 x_2 \end{aligned}$$

D'après le théorème 2.1, la plus petite solution vérifie $x_2 = (01^*0)^* 1 x_1$.
En reportant cette expression de x_2 *dans la première équation, nous obtenons*

$$x_1 = 0x_1 + 1(01^*0)^* 1 x_1 + \varepsilon$$

D'après le théorème 2.1, la plus petite solution en x_1 est donc $x_1 = (0+1(01^*0)^*1)^* + \varepsilon = (0+1(01^*0)^*1)^*$.
En reportant la valeur de x_1 dans les équations $x_2 = (01^*0)^*1x_1$ et $x_3 = 1^*0x_2$, on obtient la plus petite solution (dont on verra au chapitre IX qu'elle est unique) :

$$\begin{aligned} x_1 &= (0+1(01^*0)^*1)^* \\ x_2 &= (01^*0)^*1(0+1(01^*0)^*1)^* \\ x_3 &= 1^*0(01^*0)^*1(0+1(01^*0)^*1)^* \end{aligned}$$

3 Système d'équations associé à un automate fini

Théorème 3.1 *Soit $A =< Q, V, \delta, I, F >$ un automate fini et $q_1, \ldots, q_n$ une liste sans répétition des états de l'automate.*
Pour i de 1 à n, soit $A_i =< Q, V, \delta, \{q_i\}, F >$ l'automate identique à A sauf que son unique état initial est q_i.
Soit pour tous les i, j entre 1 tet n, $\alpha_{i,j} = \{a \in V \cup \{\varepsilon\} \mid (q_i, a, q_j) \in \delta\}$.
Soient $x_1, \ldots, x_n$ une liste de n variables distinctes.
Le système d'équations sur V défini pour i de 1 à n par

$$\begin{aligned} &\text{si } q_i \in F \quad x_i = \varepsilon + \textstyle\sum_{j=1}^n \alpha_{i,j} x_j \\ &\text{si } q_i \notin F \quad x_i = \textstyle\sum_{j=1}^n \alpha_{i,j} x_j \end{aligned}$$

a pour plus petite solution $x_1 = L(A_1), \ldots, x_n = L(A_n)$.

Preuve :

1. $x_1 = L(A_1), \ldots, x_n = L(A_n)$ est une solution du système.
   En effet $u \in L(A_i)$
   si et seulement si, par définition de A_i, il y a un chemin de trace u entre q_i et un état de F
   si et seulement si, par définition des traces,
   (a) soit $u = \varepsilon$ et $q_i \in F$
   (b) soit il y a un chemin avec au moins une transition de trace u entre q_i et un état de F
   si et seulement si, par définition des traces,
   (a) soit $u = \varepsilon$ et $q_i \in F$
   (b) soit il y a un entier j entre 1 et n, un symbole a et une chaine v tels que $u = av, a \in \alpha_{i,j}, v \in L(A_j)$
   Par suite
   $$\begin{aligned} &\text{si } q_i \in F \quad L(A_i) = \varepsilon + \textstyle\sum_{j=1}^n \alpha_{i,j} L(A_j) \\ &\text{si } q_i \notin F \quad L(A_i) = \textstyle\sum_{j=1}^n \alpha_{i,j} L(A_j) \end{aligned}$$
   Ce qui prouve la propriété 1
2. $x_1 = L(A_1), \ldots, x_n = L(A_n)$ est la plus petite solution.
   Soit $x_1 = M_1, \ldots, x_n = M_n$ une solution du système.
   Soit $L_{i,j}$ l'ensemble des chaines $u \in V^*$ telles que il y a un chemin de longueur j et de trace u entre l'état q_i et un état de F.
   Soit P(j) la propriété : pour i de 1 à n, $L_{i,j} \subset M_i$
   On montre d'abord que P(j) est vraie pour tout $j \in \mathbb{N}$.
   Par définition des $L_{i,j}$, nous avons
   $$\begin{aligned} &\text{si } q_i \in F \quad L_{i,0} = \{\varepsilon\} \\ &\text{si } q_i \notin F \quad L_{i,0} = \emptyset \end{aligned}$$
   Puisque $x_1 = M_1, \ldots, x_n = M_n$ est une solution du système, pour i de 1 à n, si $q_i \in F$ alors $\varepsilon \in M_i$.
   Par suite pour i de 1 à n, $L_{i,0} \subset M_i$.
   Ce qui prouve P(0).

Supposons P(j) vérifiée et prouvons P(j+1).
Par définition des $L_{i,j}$ et des $\alpha_{i,j}$, nous avons

$$L_{i,j+1} = \sum_{k=1}^{n} \alpha_{i,k} L_{k,j} \tag{V.1}$$

Par hypothèse de récurrence $L_{k,j} \subset M_k$, donc $\alpha_{i,k} L_{k,j} \subset \alpha_{i,k} M_k$.
De l'égalité V.1 et de cette dernière inégalité, il résulte que :

$$L_{i,j+1} \subset \sum_{k=1}^{n} \alpha_{i,k} M_k \tag{V.2}$$

Puisque $x_1 = M_1, \ldots, x_n = M_n$ est une solution du système, il en résulte que pour i de 1 à n

$$\sum_{k=1}^{n} \alpha_{i,k} M_k \subset M_i \tag{V.3}$$

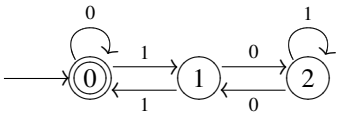
Des inégalités V.2 et V.3, il résulte que pour i de 1 à n

$$L_{i,j+1} \subset M_i$$

Ce qui montre que la propriété P(j) est vraie pour tout $j \in \mathbb{N}$.
Il nous reste à prouver que pour i de 1 à n, $L(A_i) \subset M_i$.
Soit $u \in L(A_i)$. Par définition des $L_{i,j}$, il existe j tel que $j \geq 0$ et $u \in L_{i,j}$.
D'après la propriété P(j), pour tout $i \in \mathbb{N}, L_{i,j} \subset M_i$
Donc $u \in M_i$.
Par suite $x_1 = L(A_1), \ldots, x_n = L(A_n)$ est bien la plus petite solution du système.

□

Exemple 3.1 *Soit A l'automate ci-dessous*



Cet automate déterministe reconnait les chaines binaires représentant un nombre écrit en base 2 et divisible par 3. Pour le prouver, il faut associer les états $0,1,2$ *aux nombres binaires égaux respectivement à* $0,1,2$ *modulo 3 et remarquer que le corps* $\mathbb{Z}/3\mathbb{Z}$ *on a*

- $2*0+0=0, 2*0+1=1$, *ce qui «explique» les flêches sortant de l'état* 0
- $2*1+0=2, 2*1+1=0$, *ce qui «explique» les flêches sortant de l'état* 1
- $2*2+0=1, 2*2+1=2$, *ce qui «explique» les flêches sortant de l'état* 2

D'après le théorème 3.1 on associe à cet automate le système

$$\begin{aligned} x_1 &= 0x_1 + 1x_2 + \varepsilon \\ x_2 &= 1x_1 + 0x_3 \\ x_3 &= 0x_2 + 1x_3 \end{aligned}$$

dont les variables x_1, x_2, x_3 *correspondent respectivement aux états* $0,1,2$
dont la plus petite solution vérifie $x_1 = L(A)$.
Nous avons déjà trouvé cette solution à l'exemple 2.1 donc $L(A) = (0+1(01^*0)^*1)^*$.

Théorème 3.2 *Tout langage reconnu par un automate fini est un langage régulier.*

Preuve : Soit $A =< Q, V, \delta, I, F >$ un automate fini. Soit $q_1, \ldots, q_n$ une liste sans répétition des états de l'automate.
Pour i de 1 à n, soit $A_i =< Q, V, \delta, \{q_i\}, F >$.
Par définition des automates A_i, on a : $L(A) = \sum_{i \text{ tel que } q_i \in I} L(A_i)$.
D'après le théorème 3.1, on associe à l'automate A un système régulier (ses coefficients sont des langages finis) dont

les variables correspondent respectivement aux états $q_1, \ldots, q_n$ et dont la plus petite solution est
$x_1 = L(A_1), \ldots, x_n = L(A_n)$.
D'après le théorème 2.2, le systeme étant régulier, sa plus petite solution est régulière, donc les langages $L(A_i)$ sont réguliers.
Puisque $L(A)$ est une somme finie de ces langages, $L(A)$ est un langage régulier. □

Théorème 3.3 *Un langage est reconnu par un automate fini si et seulement si il est défini par une expression régulière.*

Preuve : D'après le théorème ci-dessus 3.2, un langage reconnu par un automate est régulier.
Réciproquement d'après le théoreme 1.1, un langage régulier est reconnu par un automate. □

Chapitre VI

Automates finis déterministes minimaux

Ce chapitre est consacré à la minimisation des automates finis déterministes. Un automate déterministe est minimal si tout automate déterministe qui lui est équivalent comporte au moins autant d'états. Notre étude est divisée en deux parties

- Construction de l'automate minimal, ce qui est le résultat intéressant en pratique.
- Unicité (au nom près des états) de l'automate minimal. Comme ce résultat est intéressant en théorie, nous en donnerons deux preuves, en espérant que le lecteur en étudiera au moins une.

Ces résultats s'étendent aux machines de Mealy et de Moore (définies au paragraphe 7 du Chapitre IV) et nous laissons cette étude en exercice.
Comme dans ce chapitre, tous les automates sont finis et déterministes, nous nous abstiendrons, sauf exception, de le rappeler.

1 Construction de l'automate minimal

Définition 1.1 (Équivalence entre les états) *Soit $A =< Q, V, \delta, i, F >$ un automate. Soit p un état de cet automate. On note A_p l'automate $< Q, V, \delta, p, F >$. Deux états p et q sont équivalents si les deux automates A_p et A_q sont équivalents. On note $\equiv_A$ l'équivalence associée à l'automate A, l'indice A pouvant être omis quand il n'y a pas d'ambiguïté.*

Propriété 1.1 (De l'équivalence entre les états) *Soit $A =< Q, V, \delta, i, F >$ un automate.*

1. *Les états p et q sont équivalents si et seulement si*
   pour toute chaine $x \in V^*$, $\delta(p,x) \in F$ si et seulement si $\delta(q,x) \in F$
2. *Si les états p et q sont équivalents, alors pour toute chaine $x \in V^*$, les états $\delta(p,x)$ et $\delta(q,x)$ sont équivalents*

Preuve :

1. Les états p et q sont équivalents si et seulement si $L(A_p) = L(A_q)$.
   D'après la propriété 5.2, $L(A_p) = \{x \in V^* \mid \delta(p,x) \in F\} = L(A_q) = \{x \in V^* \mid \delta(q,x) \in F\}$.
   Par suite p et q sont équivalents si et seulement si
   pour toute chaine $x \in V^*$, $\delta(p,x) \in F$ si et seulement si $\delta(q,x) \in F$.
2. Supposons que les états p et q sont équivalents. Pour montrer que pour toute chaine $x \in V^*$, les états $\delta(p,x)$ et $\delta(q,x)$ sont équivalents, il suffit, d'après la propriété ci-dessus 1, de prouver que :
   pour toute chaine $y \in V^*$, $\delta(\delta(p,x),y) \in F$ si et seulement si $\delta(\delta(q,x),y) \in F$.
   $\delta(\delta(p,x),y) \in F$ si et seulement si, d'après la propriété 5.3 du chapitre IV
   $\delta(p,xy) \in F$ si et seulement si , sachant que p et q sont équivalents et d'après la propriété ci-dessus 1
   $\delta(q,xy) \in F$ si et seulement si, d'après la propriété 5.3 du chapitre IV
   $\delta(\delta(p,x),y) \in F$

□

Pour minimiser un automate A, on procède en deux étapes

– on enlève les états inacessibles depuis l'état initial (voir le théoreme 5.1).
– on identifie les états équivalents

Nous précisons cette deuxième étape :

– nous montrons qu'elle change un automate connecté, c'est à dire sans états inacessibles, en un automate connecté équivalent.
– nous décrivons, dans le paragraphe 2, comment calculer la relation d'équivalence entre états

Dans le paragraphe 3, nous prouvons que l'automate B ainsi obtenu est l'automate minimal équivalent à A :

– il n'y a pas d'automate équivalent avec moins d'états
– tout autre automate minimal équivalent est, au nom des états près, identique à B

Notation 1.1 *Soit ρ une relation d'équivalence sur un ensemble A et $a \in A$. La classe d'équivalence de a pour la relation ρ est notée $[a]_\rho$, l'indice pouvant être omis quand le contexte le permet.*

Définition 1.2 (Indentification des états équivalents) *Soit $A =< Q, V, \delta, i, F >$ un automate et $\equiv_A$ l'équivalence entre états associée à l'automate. On associe à A l'automate $m(A) =< R, V, \eta, [i]_{\equiv_A}, R >$ où*

– *R est l'ensemble des classes d'équivalence de la relation $\equiv_A$*
– *pour tout $p \in Q$, $a \in V$, $\eta([p]_{\equiv_A}, a) = [\delta(p,a)]_{\equiv_A}$*
– *G est l'ensemble des classes d'équivalence des états de F*

Notons que l'application η est bien définie, car, d'après la propriété 2, si les états p et q sont équivalents, il en est de même des états $\delta(p,a)$ et $\delta(q,a)$, où $a \in V$.

Propriété 1.2 (Équivalence de A et $m(A)$) *On reprend les notations ci-dessus sauf que l'on omet de rappeler l'équivalence entre états. L'automate $m(A)$ vérifie*

1. *pour tout $p \in Q$, $x \in V^*$, $\eta([p], x) = [\delta(p,x)]$*
2. *pour tout $p \in Q$, $p \in F$ si et seulement si $[p] \in G$*
3. *les automates A et $m(A)$ sont équivalents*
4. *deux états équivalents de $m(A)$, pour l'équivalence associée à $m(A)$, sont identiques.*
5. *si A est connecté alors $m(A)$ est connecté.*

Supposons que A est connecté. La propriété ci-dessus montre non seulement que les automates A et $m(A)$ sont équivalents mais aussi, qu'il est impossible, par la même construction appliquée à $m(A)$, de réduire le nombre d'états de cet automate.

Preuve :

1. Prouvons que $\eta([p], x) = [\delta(p,x)]$. On prouve la propriété par induction sur x.
   Pour $x = \varepsilon$, $\eta([p], x) = [p] = [\delta(p,x)]$ par définition des extensions de η et δ.
   Supposons que pour tout $p \in Q$, $\eta([p], x) = [\delta(p,x)]$ et soit $a \in V$. Montrons que $\eta([p], ax) = [\delta(p,ax)]$.
   $\eta([p], ax) = \eta(\eta([p], a), x)$ par définition de l'extension de η
   $\eta(\eta([p], a), x) = \eta([\delta(p,a)], x)$ par définition de η
   $\eta([\delta(p,a)], x) = [\delta(\delta(p,a), x)]$ par hypothèse d'induction
   $[\delta(\delta(p,a), x)] = [\delta(p, ax)]$ par définition de l'extension de δ
2. Montrons que $p \in F$ si et seulement si $[p] \in G$.
   Par définition de G, si $p \in F$ alors $[p] \in G$.
   Réciproquement supposons $[p] \in G$. Par définition de G, il existe un état q équivalent à p, tel que $q \in F$.
   En particulier, $\delta(q, \varepsilon) = q \in F$, et comme p est équivalent à q, $\delta(p, \varepsilon) = p$ est également dans F d'après la propriété 1.2.
3. les automates A et $m(A)$ sont équivalents
   $x \in L(A)$
   si et seulement si $\delta(i,x) \in F$ Par définition de A
   si et seulement si $[\delta(i,x)] \in G$ Par la propriété ci-dessus 2
   si et seulement si $\eta([i], x) \in G$ Par la propriété 1
   si et seulement si $x \in L(m(A))$ Par définition de $m(A)$

4. Montrons que deux états équivalents de $m(A)$, pour l'équivalence associée à $m(A)$, sont identiques.
   Soient p et q deux états de A.
   $[p]$ et $[q]$ sont équivalents pour $m(A)$
   si et seulement si pour tout $x \in V^*$, $\eta([p],x) \in G$ si et seulement si $\eta([q],x) \in G$ d'après la propriété 1.1 (1)
   si et seulement si pour tout $x \in V^*$, $[\delta(p,x)] \in G$ si et seulement si $[\delta(q,x)] \in G$ d'après la propriété 1
   si et seulement si pour tout $x \in V^*$,$\delta(p,x) \in F$ si et seulement si $\delta(q,x) \in F$ d'après la propriété 2
   si et seulement si pour tout $x \in V^*$, $[p] = [q]$ d'après la propriété 1.1 (1)
5. Supposons A connecté, montrons qu'il en est de même de $m(A)$.
   Soit $[p]$ où $p \in Q$, un état *quelconque* de $m(A)$
   Puisque A est connecté, il existe $x \in V^*$ tel que $\delta(i,x) = p$
   D'après la propriété 1, $\eta([i],x) = [p]$, donc $[p]$ est accessible.

□

2 Calcul de la relation d'équivalence entre les états d'un automate

Soit A un automate. La construction de l'automate minimal équivalent à A a été décrite au paragraphe précédent. Pour achever cette construction, nous montrons comment calculer la relation d'équivalence entre les états de A. Pour calculer cette relation, on en construit une suite d'approximations.

Définition 2.1 (Approximations de l'équivalence entre états) *Soit $A =< Q,V,\delta,i,F >$ un automate.*
Pour tout $k \in \mathbb{N}$, la relation $\equiv_k$ est définie ainsi :
pour tous $p,q \in Q$, $p \equiv_k q$ si et seulement si pour toute chaine $x \in V^*$ de longueur au plus *k, $\delta(p,x) \in F$ équivaut à $\delta(q,x) \in F$.*

Il est évident que la relation$\equiv_k$ est une relation d'équivalence.

Propriété 2.1 (Relation entre les approximations de l'équivalence $\equiv$) *Soit A le même automate que ci-dessus.*

1. $\equiv = \bigcap_{k\in\mathbb{N}} \equiv_k$.
2. *Pour tout $k \in \mathbb{N}, \equiv_{k+1} \subset \equiv_k$*
3. *Pour tous $p,q \in Q$, $p \equiv_0 q$ si et seulement si $p,q \in F$ ou $p,q \notin F$*
4. *Pour tout $k \in \mathbb{N}, p,q \in Q$, $p \equiv_{k+1} q$ si et seulement si ($p \equiv_k q$ et pour tout $a \in V, \delta(p,a) \equiv_k \delta(q,a)$)*
5. *Soit n le nombre d'états de l'automate A, il existe un entier k tel que $k \leq n$ et $\equiv_k = \equiv_{k+1} = \equiv$*

Preuve : Les trois premières propriétés sont évidentes et nous montrons seulement les deux dernières.

– Prouvons que $p \equiv_{k+1} q$ si et seulement si ($p \equiv_k q$ et pour tout $a \in V, \delta(p,a) \equiv_k \delta(q,a)$).

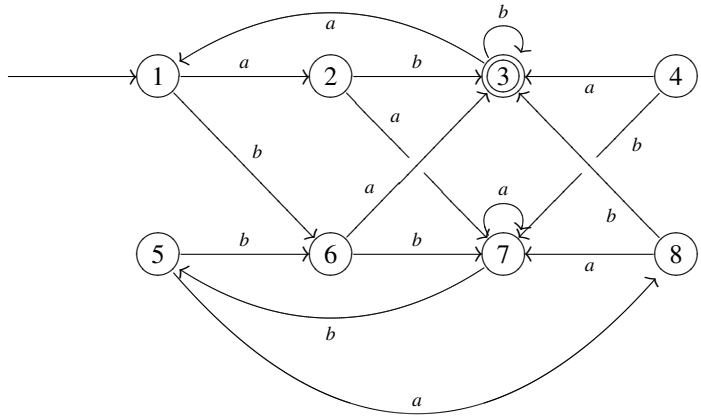
  1. Prouvons l'implication de gauche à droite. Soient p et q deux états tels que $p \equiv_{k+1} q$.
     Puisque $\equiv_{k+1} \subset \equiv_k$, on a $p \equiv_k q$.
     Par définition de la relation $\equiv_{k+1}$, on a : pour tout $a \in V$, pour toute chaine $x \in V^*$ de longueur au plus k, $\delta(p,ax) \in F$ si et seulement si $\delta(q,ax) \in F$.
     Ce qui s'écrit encore : pour tout $a \in V$, pour toute chaine $x \in V^*$ de longueur au plus k, $\delta(\delta(p,a),x) \in F$ si et seulement si $\delta(\delta(q,a),x) \in F$.
     Donc par définition de $\equiv_k$, pour tout $a \in V$, $\delta(p,a) \equiv_k \delta(q,a)$.
  2. Prouvons l'implication de droite à gauche. Soient p et q deux états tels que ($p \equiv_k q$ et pour tout $a \in V, \delta(p,a) \equiv_k \delta(q,a)$).
     De la deuxième hypothèse on déduit :
     Pour tout $a \in V$, pour toute chaine $x \in V^*$ de longueur k, $\delta(\delta(p,a),x) \in F$ équivaut à $\delta(\delta(q,a),x) \in F$.
     Cette dernière propriété se réécrit :
     Pour toute chaine $x \in V^*$ de longueur $k+1$, $\delta(p,x) \in F$ équivaut à $\delta(q,x) \in F$.
     Puisque $p \equiv_k q$, il en résulte que $p \equiv_{k+1} q$.

– Soit n le nombre d'états de l'automate A, montrons qu'il existe un entier k tel que $k \leq n$ et $\equiv_k = \equiv_{k+1} = \equiv$.

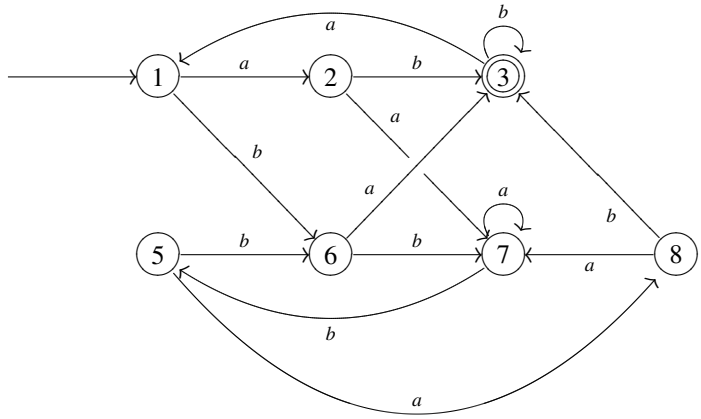
1. Il existe un entier $k \leq n$ tel que $\equiv_k = \equiv_{k+1}$.
   En effet les partitions associées aux relations d'équivalences $\equiv_k$ sont de plus en plus fines lorsque k augmente. Puisque toute partition comprend au plus n classes, il existe un entier $k \leq n$ tel que les partitions associées aux relations $\equiv_k$ et $\equiv_{k+1}$ sont égales, donc tel que ces relations sont égales.
2. Si $\equiv_m = \equiv_{m+1}$ alors $\equiv_m = \equiv$.
   Supposons que $\equiv_m = \equiv_{m+1}$. De cette hypothèse et de la propriété 4, il résulte immédiatement que pour tout $k \geq m$, on a $\equiv_k = \equiv_m$.
   Puisque la suite des relations $\equiv_k$ est une suite décroissante, stable à partir de l'indice m, il en résulte que : $\equiv_m = \bigcap_{k \in \mathbb{N}} \equiv_k$.
   Donc d'après la propriété 1, $\equiv_m = \equiv$

□

Exemple 2.1 *Minimisons l'automate déterministe A de vocabulaire $\{a,b\}$, dessiné ci-dessous :*



On remarque que l'état 4 est inaccessible. On le supprime et on obtient l'automate connecté B ci-dessous :



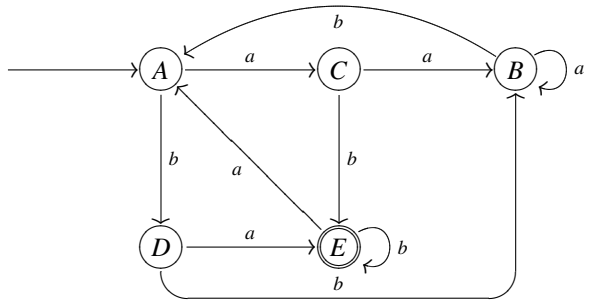
On calcule les relations $\equiv_k$ associées à l'automate B. Les numéros de la première colonne du tableau ci-dessous, sont les indices des relations d'équivalence $\equiv_k$.
D'après la propriété 4, les deux états p et q sont dans la même classe de la relation $\equiv_{k+1}$ si et seulement si

– *ils sont dans la même classe de la relation $\equiv_k$*
– *$\delta(p,a)$ et $\delta(q,a)$ sont dans la même classe de la relation $\equiv_k$*
– *$\delta(p,b)$ et $\delta(q,b)$ sont dans la même classe de la relation $\equiv_k$*

Le tableau qui suit est une disposition de ce mode de calcul, dans lequel les classes sont désignées par des majuscules.

k	*classes*						
0	*1*	*2*	*5*	*6*	*7*	*8*	*3*
			A				*B*
a	*A*	*A*	*A*	*B*	*A*	*A*	
b	*A*	*B*	*A*	*A*	*A*	*B*	
1	*1*	*5*	*7*	*2*	*8*	*6*	*3*
		A		*B*		*C*	*D*
a	*B*	*B*	*A*	*A*	*A*		
b	*C*	*C*	*A*	*D*	*D*		
2	*1*	*5*	*7*	*2*	*8*	*6*	*3*
	A		*B*	*C*		*D*	*E*
a	*C*	*C*	*B*	*B*	*B*	*E*	*A*
b	*D*	*D*	*A*	*E*	*E*	*B*	*E*
3	*1*	*5*	*7*	*2*	*8*	*6*	*3*

Puisque les partitions associées aux relations $\equiv_2$ et $\equiv_3$ sont identiques, la relation $\equiv$ est identique à $\equiv_2$. Nous pouvons maintenant construire l'automate $m(B)$, qui est dessiné ci-dessous.
Pour calculer la partition associée à $\equiv_3$, il est inutile de calculer les classes des états $\delta(7,a),\delta(7,b),\delta(6,a),\delta(6,b)$, $\delta(3,a),\delta(3,b)$ puisque les états $7,6,3$ sont seuls dans leurs classes. Mais on a fait figurer ce calcul dans la table ci-dessus, car cela fait apparaitre la fonction de transition de l'automate minimal dans les trois avant dernières lignes de ce tableau.



3 Unicité de l'automate minimal

Définition 3.1 (Automate réduit) *Un automate* réduit *est un automate dont les états sont deux à deux non équivalents.*

Nous avons vu au paragraphe 1 comment construire un automate connecté et réduit équivalent à un automate A :
- on enlève de A ses états inaccessibles, ce qui donne l'automate B
- l'automate $m(B)$ est un automate connecté et réduit équivalent à A

Dans ce paragraphe, on montre, par deux preuves bien différentes, les résultats suivants
- un automate est minimal si et seulement si il est connecté et réduit (voir théorème 3.1.
  Il en résulte que $m(B)$ est un automate minimal équivalent à A.
- l'automate minimal est unique, au nom près de ses états (voit théorème 3.2.
  Il en résulte que $m(B)$ est l'automate minimal équivalent à A.

3.1 Propriétés de la catégorie des automates

Propriété 3.1 *Si un automate est minimal, il est connecté et réduit*

Preuve : Supposons que l'automate A ne soit pas connecté ou pas réduit. Examinons ces deux cas.

1. Supposons que A ne soit pas connecté. En enlevant ses états inaccessibles, on obtient un automate équivalent avec moins d'états que A. Donc A n'est pas minimal.

2. Supposons que A ne soit pas réduit. L'automate $m(A)$ est un automate équivalent avec moins d'états que A. Donc A n'est pas minimal.

□

Définition 3.2 (homorphisme et isomorphisme d'automates) *Soient $A =< Q,V,\delta,i,F >$ et $B =< R,V,\eta,j,G >$ deux automates. L'application $h : Q \mapsto R$ est un homorphisme de A dans B lorsqu'elle vérifie*

- $h(i) = j$
- $h(F) = G$ *où* $h(F) = \{h(p) \mid p \in F\}$.
- *pour tout* $p \in Q, a \in V$, $h(\delta(p,a)) = \eta(h(p),a)$

Un homomorphisme bijectif est un isomorphisme.
Deux automates sont isomorphes, s'il y a un isomorphisme de l'un dans l'autre. Informellement, ces deux automates sont identiques au nom près de leurs états.

Lemme 3.1 *Soient $A =< Q,V,\delta,i,F >$ et $B =< R,V,\eta,j,G >$ deux automates équivalents et connectés. Si B est réduit alors il y a un homorphisme h de A dans B tel que $h(Q) = R$ où $h(Q) = \{h(p) \mid p \in Q\}$.*

De ce lemme, il est facile de déduire la propriété «un automate est minimal si et seulement il est connecté et réduit» ainsi que l'unicité de l'automate minimal.
Preuve : Soit $h \subset Q \times R$ la relation définie par $h = \{(\delta(i,x),\eta(j,x)) \mid x \in V^*\}$.
On montre que h est un homomorphisme de A dans B.

1. Q est le domaine de la relation h.
   En effet soit $p \in Q$. Puisque A est connecté, il existe une chaine $x \in V^*$ telle que $p = \delta(i,x)$. Donc p est dans le domaine de h.
2. h est une fonction de Q dans R telle que pour tout $x \in V^*$, $h(\delta(i,x)) = \eta(j,x)$.
   D'après la définition de la relation h et le fait, prouvé ci-dessus, que Q est le domaine de la relation h, il suffit de vérifier que :
   (a) pour toutes chaines $x,y \in V^*$, si $\delta(i,x) = \delta(i,y)$ alors $\eta(j,x) = \eta(j,y)$
   Supposons au contraire qu'il y a deux chaines $x,y \in V^*$ telles que :
   (b) $\delta(i,x) = \delta(i,y)$ et $\eta(j,x) \neq \eta(j,y)$
   Puisque l'automate B est réduit, les deux états $\eta(j,x)$ et $\eta(j,y)$ ne sont pas équivalents, donc :
   (c) il existe $z \in V^*$ tel que l'un des deux états $\eta(\eta(j,x),z)$ et $\eta(\eta(j,y),z)$ est dans G tandis que l'autre n'est pas dans G.
   Soit L le langage reconnu par les deux automates équivalents. Puisque G est l'ensemble des états finals de B, la propriété ci-dessus implique :
   (d) il existe $z \in V^*$ tel que l'une des deux chaines xz et yz est dans L tandis que l'autre n'est pas dans L.
   Puisque $\delta(i,x) = \delta(i,y)$, alors pour toute chaine $z \in V^*$, $\delta(i,xz) = \delta(i,yz)$. Puisque l'automate A reconnait L, on en déduit :
   (e) pour toute chaine $z \in V^*$, les deux chaines xz et yz sont toutes les deux éléments L, ou toutes les deux non éléments L.
   Les propositions **(d)** et **(e)** sont contradictoires, donc la proposition **(b)** est fausse et par suite la proposition **(a)** est vraie.
3. $h(Q) = R$.
   Puisque h est une application de Q dans R, il suffit de vérifier que tout élément de R est l'image par h d'un élément de Q.
   En effet soit $p \in R$. Puisque B est connecté, il existe $x \in V^*$ tel que $p = \eta(j,x)$. D'après la propriété ci-dessus 2, $p = h(\delta(i,x))$. Donc $p \in h(Q)$.

Maintenant que l'on sait que h est une fonction de Q dans R telle que

1. pour tout $x \in V^*$, $h(\delta(i,x)) = \eta(j,x)$.
2. $h(Q) = R$.

il reste à démontrer que h est un homorphisme de A dans B.

1. $h(i) = j$.
   D'après la propriété rappelée ci-dessus 1, $h(\delta(i,\varepsilon)) = \eta(j,\varepsilon)$. Par définition des extensions des fonctions δ et η, $\delta(i,\varepsilon) = i$ et $\eta(j,\varepsilon) = j$. Donc $h(i) = j$.
2. $h(F) = G$.
   On prouve que $h(F) \subset G$, puis que $G \subset h(F)$.
   (a) $h(F) \subset G$.
   Soit $p \in F$. Puisque A est connecté, il existe $x \in V^*$ tel que $p = \delta(i,x)$.
   Puisque $p \in F$, la chaine x est dans le langage reconnu par A.
   D'après la propriété 1, $h(p) = \eta(j,x)$. Puisque les automates A et B sont équivalents, x est aussi reconnu par B, donc $\eta(j,x) \in G$ et par suite $h(p) \in G$.
   (b) $G \subset h(F)$.
   Soit $p \in G$. Puisque B est connecté, il existe $x \in V^*$ tel que $p = \eta(j,x)$.
   Puisque $p \in G$, la chaine x est dans le langage reconnu par B.
   D'après la propriété 1, $p = h(\delta(i,x))$. Puisque les automates A et B sont équivalents, x est aussi reconnu par A, donc $\delta(i,x) \in F$ et par suite $p \in h(F)$
3. Pour tout $p \in Q, a \in V$, $h(\delta(p,a)) = \eta(h(p),a)$.
   Soit $p \in Q$ et $a \in V$. Puisque A est connecté, il existe $x \in V^*$ tel que $p = \delta(i,x)$.
   D'après les propriétés des automates déterministes, $h(\delta(p,a)) = h(\delta(i,xa))$.
   D'après la propriété 1, $h(\delta(i,xa)) = \eta(j,xa)$.
   D'après les propriétés des automates déterministes,$\eta(j,xa) = \eta(\eta(j,x),a)$.
   D'après la propriété 1, $\eta(\eta(j,x),a) = \eta(h(\delta(i,x)),a)$.
   Puisque $p = \delta(i,x)$, on a bien $h(\delta(p,a)) = \eta(h(p),a)$.

□

Théorème 3.1 (Propriété caractéristique de l'automate minimal) *Un automate est minimal si et seulement si il est connecté et réduit.*

Preuve : L'implication de gauche à droite est déjà prouvée par la propriété 3.1.
Réciproquement, soit $A =< Q,V,\delta,i,F >$ un automate connecté et réduit. Montrons qu'il est minimal, c'est-à-dire que tout autre automate équivalent a au moins autant d'états.
Soit $B =< R,V,\eta,j,G >$ un automate quelconque équivalent à A et soit $C =< S,V,\theta,j,H >$ l'automate obtenu en enlevant les états inaccessibles de A.
L'automate C est équivalent à A et connecté.
D'après le lemme 3.1, il y a un homorphisme h de C dans A tel que $h(S) = Q$. Par suite :
nombre d'états de $C \geq$ nombre d'états de A.
Puisque C est obtenu en enlevant les états inaccessibles de B :
nombre d'états de $B \geq$ nombre d'états de C.
Donc tout automate équivalent à A possède au moins autant d'états que A : A est minimal. □

Théorème 3.2 (Unicité de l'automate minimal) *Si deux automates sont équivalents et minimaux, ils sont isomorphes*

Preuve : Soient $A =< Q,V,\delta,i,F >$ et $B =< R,V,\eta,j,G >$ deux automates équivalents et minimaux.
Puisqu'ils sont minimaux, d'après le théorème 3.1, ils sont connectés et réduits.
Donc, d'après le lemme 3.1, il y a un homomorphisme h de A dans B tel que $h(Q) = R$.
Puisque les deux automates sont minimaux, les ensembles Q et R ont le même nombre d'éléments et puisque $h(Q) = R$, la fonction h est bijective. Donc A et B sont isomorphes. □

4 Automate minimal et théorème de Myhill-Nérode

Ce paragraphe est dédié à une preuve alternative du résultat déjà prouvé suivant :
Soit A un automate connecté. L'automate $m(A)$, défini en 1.2, est *l'unique* automate minimal équivalent à A.

Définition 4.1 (Dérivé et Équivalence entre chaînes) *Soit L un langage sur le vocabulaire V.*

1. *Soit $u \in V^*$. Soit $u^{-1}L = \{v \in V^* \mid uv \in L\}$. Ce langage est la dérivée de u relativement à L.*
2. *Deux chaînes u et v, où $u, v \in V^*$, sont équivalentes relativement à L si et seulement elles ont la même dérivée relativement à L. On note $\equiv_L$ cette relation d'équivalence entre chaînes. L'indice L peut être omis quand il n'y a pas d'ambiguïté.*

Propriété 4.1 (Propriété de l'équivalence entre chaînes) *Soit L un langage sur le vocabulaire V. Soient $u, v \in V^*$*

1. *$u \equiv_L v$ si et seulement si pour toute chaîne $x \in V^*$, $ux \in L$ si et seulement si $vx \in L$.*
2. *si $u \equiv_L v$ alors pour toute chaîne $x \in V^*$, $ux \equiv_L vx$*

La preuve est immédiate, comme pourra le vérifier le lecteur.

Définition 4.2 (Automate des classes d'un langage) *Soit L un langage sur le vocabulaire V. Soit $A(L) =< Q, V, \delta, i, F >$ l'automate déterministe suivant*[1] *:*

1. *Q est l'ensemble des classes de la relation $\equiv_L$*
2. *Pour tout $x \in V^*, a \in V$, $\delta([x]_{\equiv_L}, a) = [xa]_{\equiv_L}$*
3. *$i = [\varepsilon]_{\equiv_L}$*
4. *F est l'ensemble des classes des chaînes du langage L*

$A(L)$ est l'automate des classes du langage L.

Notons que l'application δ est bien définie, car d'après la propriété 4.1, pour tout $x, y \in V^*, a \in V$, si $x \equiv_L y$ alors $xa \equiv_L ya$.

Propriété 4.2 *Soit $A(L)$ l'automate des classes du langage L. On reprend les notations ci-dessus.*

1. *Pour tous $x, y \in V^*$, $\delta([x]_{\equiv_L}, y) = [xy]_{\equiv_L}$*
2. *L'automate $A(L)$ reconnait L.*

Preuve : On reprend les notations ci-dessus.

1. On prouve la première partie de la propriété par une induction immédiate sur y
2. Montrons la deuxième propriété :
   x est reconnu par $A(L)$
   si et seulement si $\delta([\varepsilon]_{\equiv_L}, x) \in F$
   si et seulement si $[x]_{\equiv_L} \in F$ d'après la première partie de la propriété
   si et seulement si $x \in L$ car F est l'ensemble des classes des chaînes du langage L

□

Théorème 4.1 (Théorème de Myhill-Nérode) *Le langage L est régulier si et seulement si le nombre de classes de $\equiv_L$ est fini.*

Preuve :

1. Supposons L régulier. Montrons que le nombre de classes de $\equiv_L$ est fini.
   Puisque L est régulier, il y a un automate déterministe fini $A =< Q, V, \delta, i, F >$ qui reconnait L.
   Soit h l'application suivante :
   – son domaine est l'ensemble des classes d'équivalence de la relation $\equiv_L$, définie en 4.1
   – sa portée est l'ensemble des classes d'équivalence de la relation $\equiv_A$, définie en 1.1.
   – Pour tout $x \in V^*$, $h([x]_{\equiv_L}) = [\delta(i, x)]_{\equiv_A}$

[1] le nombre des états de l'automate n'est pas nécessairement fini

(a) Montrons que $x \equiv_L y$ si et seulement si $\delta(i,x) \equiv_A \delta(i,y)$.
En effet

$x \equiv_L y$ si et seulement si (pour tout $z \in V^*$, $xz \in L$ si et seulement si $yz \in L$)	D'après la propriété 4.1
si et seulement si (pour tout $z \in V^*$,$\delta(i,xz) \in F$ si et seulement si $\delta(i,yz) \in F$	Car l'automate A reconnait L
si et seulement si (pour tout $z \in V^*$,$\delta(\delta(i,x),z) \in F$ si et seulement si $\delta(\delta(i,y),z) \in F$)	D'après 5.3, chapitre IV
si et seulement si $\delta(i,x) \equiv_A \delta(i,y)$	Par définition de $\equiv_A$

(b) h est une application entre classes.
En effet d'après la propriété 1a, si $x \equiv_L y$ alors $\delta(i,x) \equiv_A \delta(i,y)$.

(c) h est injective.
Supposons $[\delta(i,x)]_{\equiv_A} = [\delta(i,y)]_{\equiv_A}$. D'après la propriété 1a, $x \equiv_L y$, donc $[x]_{\equiv_L} = [y]_{\equiv_L}$.

Puisque h est une injection, le nombre de classes de $\equiv_L$ est au plus égal au nombre de classes de $\equiv_A$.
Il est trivial que le nombre de classes de $\equiv_A$ est au plus égal au nombre d'états de A.
Puisque A est un automate *fini*, le nombre de classes de $\equiv_L$ est donc fini.

2. Supposons que le nombre de classes de $\equiv_L$ est fini.
L'automate $A(L)$, défini en 4.2, est alors un automate déterministe *fini*, qui, d'après la propriété 4.2, reconnait L, par suite L est un langage régulier.

□

Nous redémontrons, par une autre méthode, l'unicité de l'automate minimal, déjà prouvé par le théorème 3.2.

Théorème 4.2 (L'automate minimal est unique) *Soit L un langage régulier sur le vocabulaire V. L'automate $A(L)$ des classes de L est minimal et tout automate minimal équivalent lui est isomorphe.*

Preuve :

1. $A(L)$ est minimal.
Soit A un automate quelconque reconnaissant L. Comme nous l'avons vu dans la démonstration du théorème de Myhill-Nérode, le nombre d'états de $A(L)$ est au plus égal à celui de A, donc $A(L)$ est minimal.

2. Tout automate minimal reconnaissant L est isomorphe à $A(L)$.
Soit $A =< Q,V,\delta,i,F >$ un automate minimal reconnaissant L et soit $A(L) =< R,V,\eta,j,G >$.
D'après la démonstration du théorème de Myhill-Nérode, le nombre d'états de $A(L)$ est au plus égal à celui des classes de $\equiv_A$, et trivialement le nombre de classes de $\equiv_A$ est au plus égal au nombre d'états de A.
Puisque A est minimal, le nombre d'états de $A(L)$, le nombre de classes de $\equiv_A$ et le nombre d'états de A sont égaux. Par suite il n'y a qu'un état par classes de $\equiv_A$, autrement dit A est réduit.
L'automate A est aussi connecté, car s'il ne l'était pas, il serait équivalent à l'automate obtenu en lui enlevant ses états inaccessibles, donc A ne serait pas minimal.
Soit h l'application, définie dans la preuve du théorème de Mihill-Nérode, par :
Pour toute chaîne $x \in V^*$, $h([x]_{\equiv_L}) = [\delta(i,x)]_{\equiv_A}$.
D'après cette preuve, h est une injection entre les états de $A(L)$ et les classes de $\equiv_A$. Puisque le nombre des états de $A(L)$ et le nombre des classes de classes de $\equiv_A$ sont égaux, h est une bijection.
Soit h' l'application entre les états de $A(L)$ et ceux de A définie par :
Pour toute chaîne $x \in V^*$, $h'([x]_{\equiv_L}) = \delta(i,x)$
Puisque h est une bijection et qu'il n'y a qu'un état par classe de $\equiv_A$, h' est aussi une bijection. Montrons que c'est un isomorphisme entre $A(L)$ et A. Il suffit de vérifier que c'est un homorphisme (voir la définition 3.2).

(a) $h'(j) = i$.
En effet, par définition de $A(L)$, $j = [\varepsilon]_{\equiv_L}$.
Par définition de h', $h'([\varepsilon]_{\equiv_L}) = \delta(i,\varepsilon)$.
Par définition de l'extension de δ, $\delta(i,\varepsilon) = i$. Par suite $h'(j) = i$.

(b) Pour tout $x \in V^*, a \in V$, $h'(\eta([x]_{\equiv_L},a)) = \delta(h'([x]_{\equiv_L},a)$.

$h'(\eta([x]_{\equiv_L},a)) = h'([xa]_{\equiv_L})$	D'après la propriété 4.2
$h'([xa]_{\equiv_L}) = \delta(i,xa)$	Par définition de h'
$\delta(i,xa) = \delta(\delta(i,x),a)$	Par la propriété 5.3 du chapitre IV
$\delta(\delta(i,x),a) = \delta(h'([x]_{\equiv_L}),a)$	Par définition de h'

(c) $h'(G) = F$.

i. $h'(G) \subset F$
Soit p un état de G. Il existe $x \in L$ tel que $p = [x]_{\equiv_L}$.
Par définition de h', $h'(p) = h'([x]_{\equiv_L}) = \delta(i,x)$.
Puisque $x \in L$ et que A reconnait L, $\delta(i,x) \in F$, donc $h'(p) \in F$.

ii. $F \subset h'(G)$
Soit $p \in F$. Puisque A est connecté, il existe x tel que $\delta(i,x) = p$.
Puisque p est un état final de l'automate A reconnaissant L, $x \in L$.
Par définition de h', $p = h'([x]_{\equiv_L})$. Puisque $x \in L$, $[x]_{\equiv_L}$ est un état final de $A(L)$, donc $p \in h'(G)$.

□

Nous justifions, par une autre méthode, la construction de l'automate minimal fournie au paragraphe 1.

Théorème 4.3 *Si A est un automate connecté, alors $m(A)$ est l'automate minimal équivalent à A.*

Preuve : Soit A un automate connecté et L le langage reconnu par A. Soit $m(A) =< Q,V,\delta,i,F >$. On sait que $m(A)$ est réduit et connecté. On en déduit que $m(A)$ est isomorphe à $A(L)$.
Soit h l'application entre les états de $A(L)$ et les classes de la relation $\equiv_{m(A)}$ définie par :
Pour tout $x \in V^*$, $h([x]_{\equiv_L}) = [\delta(i,x)]_{\equiv_{m(A)}}$.
D'après la preuve du théorème de Mihill-Nérode, h est une injection. Puisque $m(A)$ est connecté, on vérifie facilement que c'est une bijection.
Soit h' l'application entre les états de $A(L)$ et les états de $m(A)$ définie par :
Pour tout $x \in V^*$, $h([x]_{\equiv_L}) = \delta(i,x)$.
Puisque h est une bijection et que $m(A)$ est réduit (autrement il n'y a qu'un état par classe de $\equiv_{m(A)}$), l'application h' est aussi une bijection entre les états de $A(L)$ et ceux de $m(A)$.
Puisque $A(L)$ est minimal, $m(A)$ est aussi minimal. □

Exercice 4.1 *Soit L un langage sur le vocabulaire V et soit $A =< Q,V,\delta,i,F >$ l'automate suivant :*
- *Les états de A sont les dérivées relativement à L*
- *Pour tous $u \in V^*, a \in V$, $\delta(u^{-1}L,a) = (ua)^{-1}L$*
- *L'état initial est L*
- *Un état est final si et seulement si il est la dérivée d'une chaîne de L*

L'automate A est l'automate des dérivées de L. Montrez que cet automate reconnait L et qu'il a le même nombre d'états que l'automate $A(L)$ des classes de L

‘

Chapitre VII

Définition de langages par induction

Il est naturel de définir des langages par des définitions inductives, prenant la forme de systèmes d'équations. Les langages définis par des systèmes d'équations sont appelés des langages algébriques et dans ce chapitre

- on montre qu'un langage est algébrique si et seulement si il est hors-contexte
- on étudie les transformations des systèmes d'équations qui en préservent l'ensemble des solutions
- on donne une condition nécessaire et suffisante, *calculable* pour qu'un système d'équation ait une et une seule solution.

1 Exemples de définitions inductives de langages

Nous utilisons ci-dessous, sauf exception, les notations des expressions régulières.

1. $X = \{a^n b^n \mid n \geq 0\}$ est solution de l'équation $X = \varepsilon + aXb$. C'est aussi la plus petite solution de cette équation.
   En effet, puisque $\{\varepsilon\} \subset X$, ε est dans toute solution, et si $a^n b^n$ est dans une solution, puisque $aXb \subset X$, $a^{n+1} b^{n+1}$ est aussi dans cette solution.
   Donc, par induction sur n, iv$\{a^n b^n \mid n \geq 0\}$ est un sous-ensemble de toute solution.
   Cette équation n'a d'ailleurs qu'une solution. Supposons au contraire qu'il y a un plus petit mot x dans une solution L, avec $x \notin \{a^n b^n \mid n \geq 0\}$.
   Puisque x est un mot de la solution L, il s'écrit ayb avec $y \in L$ et $y \notin \{a^n b^n \mid n \geq 0\}$, ce qui contredit le fait que x est le plus petit mot avec cette propriété.
   Notons que le langage $\{a^n b^n \mid n \geq 0\}$ est engendré par la grammaire $X \to \varepsilon | aXb$: le signe égal de l'équation devient une flêche et le plus devient une barre.
2. $X = \{a^p b^q c^{p+q} \mid p \geq 0, q \geq 0\}$ est solution de l'équation $X = \{b^q c^q \mid q \geq 0\} + aXc$. C'est aussi la plus petite.
   En effet, puisque $\{b^q c^q \mid q \geq 0\} \subset X$, $\{b^q c^q \mid q \geq 0\}$ est dans toute solution, et si $a^p b^q c^{p+q}$ est dans une solution, puisque $aXc \subset X$, $a^{p+1} b^q c^{p+q+1}$ est aussi dans cette solution.
   Donc, par induction sur p, $\{a^p b^q c^{p+q} \mid p \geq 0, q \geq 0\}$ est un sous-ensemble de toute solution.
   Puisque ci-dessus, on a vu que $\{b^n c^n \mid n \geq 0\}$ est la plus petite solution de $Y = \varepsilon + bYc$, il en résulte que $\{a^p b^q c^r \mid p + q = r\}$ est la valeur de X de la plus petite solution du système :
   $X = Y + aXc$
   $Y = \varepsilon + bYc$
   Comme ci-dessus, on peut vérifier que cette solution est unique.
   Notons que le langage $\{a^p b^q c^{p+q} \mid p \geq 0, q \geq 0\}$ est engendré par le non-terminal X de la grammaire
   $X \to Y | aXc$
   $Y \to \varepsilon | bYc$
   Comme ci-dessus, le signe égal des équations devient une flêche et le plus devient une barre.
3. On définit les expressions dont les opérandes sont a et b et les opérateurs $+$ et $*$.
   Toute expression est un terme, ou bien s'écrit $e + t$ où e est une expression et t est un terme.
   Tout terme est un facteur, ou bien s'écrit $t * f$ où t est un terme et f est un facteur.

Un facteur est de la forme a, b ou (e) où e est une expression.
D'après la définition ci-dessus, l'ensemble des expressions, termes et facteurs est la plus petite solution respectivement en E, T et F du système d'équations :
$E = T + E \underline{+} T$
$T = F + T * F$
$F = a + b + (E)$
Notons que le signe $\underline{+}$ des expressions est souligné pour le distinguer du plus des expressions régulières.
Comme dans les cas ci-dessus, la solution est unique.

2 Théorèmes du point fixe

Toute définition inductive peut être vue comme la solution d'une équation $x = f(x)$ où $f : E \mapsto E$, autrement dit comme un point fixe de la fonction f. Aussi on étudie les conditions d'existence et de construction de points fixes.

2.1 Ordre faiblement complet

Définition 2.1 (ordre faiblement complet) *Soit E un ensemble ordonné.*
Une suite dénombrable sur cet ensemble, est une suite u_i d'éléments de E où $i \in \mathbb{N}$.
Une relation d'ordre sur un ensemble E est dite faiblement complète *si toute suite croissante dénombrable a une borne supérieure. Un ensemble ordonné est faiblement complet, si sa relation d'ordre est faiblement complète.*

Lorqu'elle existe, la borne supérieure de l'ensemble X est notée $sup(X)$.
Lorsqu'elle existe, la borne inférieure de l'ensemble X est notée $inf(X)$.

Définition 2.2 (fonction continue) *Soient E et F deux ensembles faiblement complets. La fonction $f : E \mapsto F$ est continue, si pour toute suite u_i croissante dénombrable d'éléments de E, $f(sup\{u_i \mid i \in \mathbb{N}\}) = sup\{f(u_i) \mid i \in \mathbb{N}\}$*

Propriété 2.1 *Toute fonction continue est croissante*

Preuve : Soient E et F deux ensembles faiblement complets et $f : E \mapsto F$ une fonction continue. Soient $a, b \in E$ avec $a \leq b$. Soit u_i la suite dénombrable où $u_0 = a$ et pour $i \in \mathbb{N}$, $u_{i+1} = b$.
Puisque $a \leq b$, $sup\{u_i \mid i \in \mathbb{N}\} = b$, donc par continuité $f(b) = sup\{f(a), f(b)\}$, autrement dit $f(a) \leq f(b)$. □

Théorème 2.1 (Théorème de Kleene) *Soit E un ensemble faiblement complet ayant un plus petit élément noté $\perp$. Soit f une fonction continue de E dans lui-même. Le plus petit point fixe de f est $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$.*

Preuve :

1. La suite $f^i(\perp)$ où $i \in \mathbb{N}$ est croissante et donc, $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$ est bien définie.
   Montrons, par induction sur i, que pour tout $i \in \mathbb{N}$, $f^i(\perp) \leq f^{i+1}(\perp)$.
   Puisque $\perp$ est le plus petit élément de E, $\perp \leq f(\perp)$, ce qui rend vraie la propriété pour $i = 0$.
   Supposons $f^i(\perp) \leq f^{i+1}(\perp)$. Puisque f est continue, elle est croissante donc, en appliquant f aux deux membres de l'inégalité ci-dessus, $f^{i+1}(\perp) \leq f^{i+2}(\perp)$.
   Puisque E est faiblement complet et que la suite $f^i(\perp)$ est croissante, $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$ existe.
2. $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$ est un point fixe de f.
   Puisque $\perp$ est le plus petit élément de E, on a :
   (a) $sup\{f^i(\perp) \mid i \in \mathbb{N}\} = sup\{f^{i+1}(\perp) \mid i \in \mathbb{N}\}$.
   Par continuité, $f(sup\{f^i(\perp) \mid i \in \mathbb{N}\}) = sup\{f^{i+1}(\perp) \mid i \in \mathbb{N}\}$.
   D'après l'égalité (a), $f(sup\{f^i(\perp) \mid i \in \mathbb{N}\}) = sup\{f^i(\perp) \mid i \in \mathbb{N}\}$.
3. $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$ est le plus petit des points fixes de f.
   Soit a un point fixe de f. Il suffit de montrer que pour tout $i \in \mathbb{N}$, $f^i(\perp) \leq a$.
   Pour $i = 0$, cette inégalité est vraie, car $\perp$ est le plus petit élément de E.
   Supposons $f^i(\perp) \leq a$. Puisque f est croissante, $f^{i+1}(\perp) \leq f(a)$. Puisque a est un point fixe, $f^{i+1}(\perp) \leq a$.
   Donc par induction, pour tout $i \in \mathbb{N}$, $f^i(\perp) \leq a$, et par suite $sup\{f^i(\perp) \mid i \in \mathbb{N}\} \leq a$

□

2.2 Ordre complet

Définition 2.3 (ordre complet, treillis complet) *Soit E un ensemble ordonné.*
Une relation d'ordre sur un ensemble E est dite complète *si tout sous-ensemble de E a une borne supérieure dans E.*
Un ensemble ordonné est complet, si sa relation d'ordre est complète.
Un ensemble ordonné complet est aussi appelé un treillis *complet.*

De la définition d'un treillis complet E, il résulte que E a un plus grand élément $sup(E)$ et un plus petit élément $sup(\emptyset)$.

Exemple 2.1

1. *Soit E un ensemble. L'ensemble $\mathcal{P}(E)$,aussi noté 2^E, ordonné par $\subset$ est un treillis complet. En effet soit X un sous-ensemble de $\mathcal{P}(E)$, la borne supérieure de X est l'ensemble $\bigcup X$.*
2. *Soient X et E deux ensembles. Soit $\mathcal{P}(E)^X$ l'ensemble des applications X dans $\mathcal{P}(E)$. On ordonne cet ensemble par l'extension de l'ordre des sous-ensembles de E aux fonctions :*
   Soient $f, g \in \mathcal{P}(E)^X$, $f \subset g$ si et seulement pour tout $x \in X$, $f(x) \subset g(x)$.
   Soit Y un sous-ensemble de $\mathcal{P}(E)^X$, la borne supérieure de Y est aussi notée $\bigcup Y$ et elle est définie par :
   pour tout $x \in X$, $(\bigcup Y)(x) = \bigcup\{f(x) \mid f \in Y\}$. Par suite $\mathcal{P}(E)^X$, muni de l'ordre défini ci-dessus, est un treillis complet.

Propriété 2.2 *Tout sous-ensemble d'un treillis complet a une borne inférieure.*

Preuve : Soit E un treillis complet et $X \subset E$. Soit M l'ensemble des minorants de X.
Puisque E est complet, la borne supérieure de M existe. Soit m cette borne.
Tout élément de X est un majorant de M, donc m, étant le plus petit des majorants de M, est un minorant de X. Donc $m \in M$. Puisque m est la borne supérieure de M, m est aussi le plus grand élément de M, donc la borne inférieure de X. □

Définition 2.4 (treillis) *Soit E un ensemble ordonné.*
E est un treillis si et seulement si tout sous-ensemble avec deux éléments a une borne supérieure et une borne inférieure.

Propriété 2.3 *Soit E un ensemble ordonné.*
E est un treillis si et seulement si tout sous-ensemble fini non vide a une borne supérieure et une borne inférieure.

La preuve est évidente par induction sur le nombre d'éléments des sous-ensembles finis non vides.

Remarque 2.1 *Puisque tout sous-ensemble d'un treillis complet possède une borne supérieure et une borne inférieure, un treillis complet est bien un treillis.*

Théorème 2.2 (Théorème de Tarski) *Soit E un treillis complet ordonné par $\leq$. Soit $f : E \mapsto E$ une fonction croissante. Alors $inf\{c \in E \mid f(c) \leq c\}$ est le plus petit point fixe de f.*

Preuve : Soit $C = \{c \in E \mid f(c) \leq c\}$ et $p = inf(C)$.

1. L'ensemble C n'est pas vide. En effet, s'il l'était, on aurait $f(c) > c$ pour tout $c \in E$. Donc en particulier, comme $sup(E) \in E$, on devrait avoir $f(sup(E)) > sup(E)$, ce qui est impossible.
2. Montrons que $f(p) \leq p$.
   Soit $c \in C$. Puisque $p = inf(C)$, $p \leq c$. Puisque f est croissante $f(p) \leq f(c)$. Puisque $c \in C$, $f(c) \leq c$. Par suite $f(p) \leq c$.
   L'élément $f(p)$ est donc un minorant de C, puisque p est le plus grand de ces minorants, $f(p) \leq p$.
3. Montrons que p est le plus petit point fixe de f.
   Puisque $f(p) \leq p$ et que f est croissante, $f(f(p)) \leq f(p)$. Donc $f(p) \in C$ et puisque p est la borne inférieure de C, $p \leq f(p)$.
   Donc $p = f(p)$. Puisque tous les points fixes de f sont éléments de C, et que p est la borne inférieure de C, il en résulte que p est le plus petit point fixe de f.

□

2.3 Treillis finis

1. Un treillis fini, *ayant un plus petit élément*, est complet.
   En effet soit E un treillis fini, autrement dit E est fini, ayant un plus petit élément $\perp$. Tout sous-ensemble non vide de E étant fini, il a une borne supérieure. De même $sup(\emptyset) = \perp$. Donc tout sous-ensemble de E a une borne supérieure.
2. Dans un treillis fini, une fonction croissante est continue.
   En effet soit E un treillis fini et $f : E \mapsto E$ une fonction croissante. Soit u_i une suite croissante dénombrable d'éléments de E. Puisque E est fini, il existe un plus petit entier j qui est le plus grand élément de la suite, la suite devenant stationnaire à partir de cet indice. Donc $sup\{u_i \mid i \in \mathbb{N}\} = u_j$.
   Puisque f est croissante, la suite $f(u_i)$ est aussi croissante et stationnaire à partir de l'indice j.
   Donc $sup\{f(u_i) \mid i \in \mathbb{N}\} = f(u_j)$.
   Par suite $f(sup\{u_i \mid i \in \mathbb{N}\}) = sup\{f(u_i) \mid i \in \mathbb{N}\}$, autrement dit f est continue.
3. Calcul du plus petit point fixe d'une fonction croissante dans un treillis fini, ayant un plus petit élément.
   Soit E un treillis fini, ayant un plus petit élément $\perp$. Soit $f : E \mapsto E$ une fonction croissante.
   Puisque f est continue, d'après le théorème de Kleene 2.1, le plus petit point fixe de f est $sup\{f^i(\perp) \mid i \in \mathbb{N}\}$.
   Dans la preuve de ce théorème, il est montré que la suite $f^i(\perp)$ est croissante.
   Soit n le nombre d'éléments de E. Puisque la suite $f^i(\perp)$ est croissante, il existe un plus petit indice j où $j < n$ tel que $f^j(\perp) = f^{j+1}(\perp)$.
   Une induction triviale montre que pour tout k tel que $k \geq j$, on a $f^j(\perp) = f^k(\perp)$. Donc$f^j(\perp)$ est le plus petit point fixe de f. Aussi, pour calculer ce plus petit point fixe, il suffit de calculer la suite $f^i(\perp)$ jusqu'au premier indice $j < n$ tel que $f^j(\perp) = f^{j+1}(\perp)$.

3 Langages algébriques

Définition 3.1 (Système d'équations) *Soient V_T et V_N deux vocabulaires disjoints et V l'union de ces deux vocabulaires. V_T est l'ensemble des terminaux et V_N est l'ensemble des non terminaux. Un non terminal est aussi appelé une variable.*
Un système d'équations sur V_T et V_N est un triplet $< V_T, V_N, d >$ où d *est une application de V_N dans l'ensemble des parties* finies *de $\mathcal{P}(V^*)$.*
Soit S un tel système d'équations, il est présenté sous la forme : pour toute variable A, $A = \mathrm{d}(A)$.
L'ensemble fini *de mots* $\mathrm{d}(A)$ *est la* définition *de A. Il est écrit comme la* somme *de ses mots.*

Voici comment se présente un système d'équations avec deux terminaux a, b et deux variables A_0, A_1 :

$$\begin{array}{lcl} A_0 & = & aA_0 + bA_1 \\ A_1 & = & aA_1b + \varepsilon \end{array}$$

Définition 3.2 (Interprétation) *Soient V_T et V_N deux vocabulaires disjoints et V l'union de ces deux vocabulaires. Une* interprétation *de V_N sur V_T est une application* I *de V_N dans l'ensemble des parties de $\mathcal{P}(V_T^*)$; on note* $\mathrm{I}(A)$ *l'image de la variable $A \in V_N$.*
Soit I *une telle interprétation. Elle est étendue aux mots et langages sur V de la façon suivante :*
- $\mathrm{I}(\varepsilon) = \{\varepsilon\}$
- *Si $a \in V_T$, alors* $\mathrm{I}(a) = \{a\}$
- *Si $x, y \in V^*$, alors* $\mathrm{I}(xy) = \mathrm{I}(x)\mathrm{I}(y)$
- *Si L est un langage sur V, alors* $\mathrm{I}(L) = \bigcup\{\mathrm{I}(x) \mid x \in L\}$

Exemple 3.1 *Soient a, b deux terminaux et A_0, A_1 deux variables.*
Soit I *l'interprétation suivante :* $\mathrm{I}(A_0) = \{a^nb^n \mid n \geq 0\}$ *et* $\mathrm{I}(A_1) = \{b^n \mid n \geq 0\}$.
Par définition des extensions de I*, on a :* $\mathrm{I}(bA_0 + aA_1) = \{ba^nb^n \mid n \geq 0\} \cup \{ab^n \mid n \geq 0\}$.

Remarque 3.1 *L'interprétation* I *de l'ensemble V_N de variables est l'ensemble de couples $\{(A, \mathrm{I}(A)) \mid A \in V_N\}$. Ces couples sont souvent notés $A = \mathrm{I}(A)$.*
C'est ainsi que l'interprétation I *ci-dessus est aussi notée :* $A_0 = \{a^nb^n \mid n \geq 0\}, A_1 = \{b^n \mid n \geq 0\}$

Définition 3.3 (Solution d'un système d'équations) *Soit $S =< V_T, V_N, d >$ un système d'équations.*
L'interprétation $\mathrm{I} : V_N \mapsto \mathcal{P}(V_T^*)$ *est une* solution de *S si et seulement si pour tout $A \in V_N$,* $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A))$.

Exemple 3.2 *Soit le système, avec deux terminaux a, b et deux variables A_0, A_1, déjà présenté ci-dessus :*

$$\begin{array}{lcl} A_0 & = & aA_0 + bA_1 \\ A_1 & = & aA_1b + \varepsilon \end{array}$$

L'interprétation $A_0 = \{a^n b a^p b^p \mid n, p \geq 0\}, A_1 = \{a^n b^n \mid n \geq 0\}$ est solution de S. C'est d'ailleurs l'unique solution de ce système.

Définition 3.4 (Fonction définie par un système d'équations) *Soit $S =< V_T, V_N, d >$ un système d'équations.*
La fonction définie par *S, notée f_S, est une fonction de l'ensemble $\mathcal{P}(V_T^*)^{V_N}$ des interprétations dans lui-même, définie pour toute interprétation* I *par : pour tout $A \in V_N$, $f_S(\mathrm{I})(A) = \mathrm{I}(\mathrm{d}(A))$.*

D'après les deux dernières définitions, I est solution de S si et seulement si I est un point fixe de f_S.
On rappelle que, dans ce même exemple, si E est un ensemble d'interprétations, alors l'interprétation $(\bigcup E)$ est définie par :

pour tout $A \in V_N, (\bigcup E)(A) = \bigcup\{\mathrm{I}(A) \mid \mathrm{I} \in E\}$.

D'après l'exemple 2.1, l'ensemble $\mathcal{P}(V_T^*)^{V_N}$ des interprétations ordonné par l'extension suivante de $\subset$:

Soient I, J deux interprétations, $\mathrm{I} \subset \mathrm{J}$ si et seulement si pour tout $A \in V_N, \mathrm{I}(A) \subset \mathrm{J}(A)$,

est un treillis complet. Nous nous servirons de cet ordre par la suite.

Nous allons montrer que f_S est une fonction continue, ce qui nous permettra d'utiliser le théorème de Kleene pour caractériser la plus petite solution du système S.
Avant de faire cette preuve, nous avons besoin de quelques propriétés de suites croissantes d'interprétations.

Propriété 3.1 (Suite croissante d'interprétations et concaténation) *Soient V_T et V_N deux vocabulaires disjoints et V l'union de ces deux vocabulaires. Soit E un ensemble* totalement ordonné *d'interprétations de V_N sur V_T, autrement dit un sous-ensemble totalement ordonné de $\mathcal{P}(V_T^*)^{V_N}$, avec pour ordre l'extension de $\subset$ définie ci-dessus.*
Pour tous $x, y \in V^*$, $\bigcup\{\mathrm{I}(x)\mathrm{I}(y) \mid \mathrm{I} \in E\} = (\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\})(\bigcup\{I(y) \mid \mathrm{I} \in E\})$

Preuve :

1. Soit $z \in \bigcup\{\mathrm{I}(x)\mathrm{I}(y) \mid \mathrm{I} \in E\}$.
   Montrons que $z \in (\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\})(\bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\})$.
   D'après l'hypothèse, il existe une interprétation $\mathrm{I} \in E$ telle que $z \in \mathrm{I}(x)\mathrm{I}(y)$.
   Donc $z = uv$ où $u \in \mathrm{I}(x)$ et $v \in \mathrm{I}(y)$.
   Donc $u \in \bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\}$ et $v \in \bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\}$.
   Par suite $z \in (\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\})(\bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\})$.
2. Soit $z \in (\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\})(\bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\})$.
   D'après l'hypothèse, $z = uv$ avec $u \in \bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\}$ et $v \in \bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\}$.
   Donc il existe $\mathrm{I} \in E$ telle que $u \in \mathrm{I}(x)$ et $\mathrm{J} \in E$ telle que $v \in \mathrm{J}(y)$. Puisque E est *totalement ordonné*, I et J sont comparables : soit K la plus grande de ces deux interprétations.
   On a $u \in \mathrm{K}(x)$ et $v \in \mathrm{K}(y)$, Puisque $z = uv$, il en résulte que $z \in \mathrm{K}(x)\mathrm{K}(y)$ et par suite $z \in \bigcup\{\mathrm{I}(x)\mathrm{I}(y) \mid \mathrm{I} \in E\}$.

□

Propriété 3.2 (Suite croissante d'interprétations et mots) *Soient V_T et V_N deux vocabulaires disjoints et V l'union de ces deux vocabulaires. Soit E un ensemble* totalement ordonné *d'interprétations de V_N sur V_T.*
Pour tout $x \in V^*$, $\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\} = (\bigcup E)(x)$

Preuve : On prouve la propriété par induction sur la longueur de x.

1. Si $x = \varepsilon$, alors $\mathrm{I}(x) = \{\varepsilon\}$. Par suite $\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\} = \{\varepsilon\} = (\bigcup E)(x)$
2. Si $x \in V_T$, alors $\mathrm{I}(x) = \{x\}$. Par suite $\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\} = \{x\} = (\bigcup E)(x)$

3. Soit $x \in V_N$. D'après la définition de $\bigcup E$ rappelée ci-dessus après la définition 3.4, on a : $\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\} = (\bigcup E)(x)$
4. Soit $x \in V^*$ de longueur supérieure à 1.
   Supposons que pour tout mot u de longueur inférieure à x, on ait : $\bigcup\{\mathrm{I}(u) \mid \mathrm{I} \in E\} = (\bigcup E)(u)$
   Puisque la longueur de x est supérieure à 1, il existe y, z deux chaînes non vides telles que $x = yz$. On a alors :

$$\begin{array}{rcll}
\bigcup\{\mathrm{I}(x) \mid \mathrm{I} \in E\} & = & \bigcup\{\mathrm{I}(yz) \mid \mathrm{I} \in E\} & \text{car } x = yz \\
& = & \bigcup\{\mathrm{I}(y)\mathrm{I}(z) \mid \mathrm{I} \in E\} & \text{car } \mathrm{I}(yz) = \mathrm{I}(y)\mathrm{I}(z) \\
& = & (\bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\})(\bigcup\{\mathrm{I}(z) \mid \mathrm{I} \in E\}) & \text{d'après la propriété 3.1 (car } E \text{ est totalement ordonné)} \\
& = & (\bigcup E)(y)(\bigcup E)(z) & \text{par l'hypothèse d'induction appliquée à } y \text{ et } z \\
& = & (\bigcup E)(yz) & \text{car } \bigcup E \text{ est une interprétation} \\
& = & (\bigcup E)(x) & \text{car } x = yz
\end{array}$$

□

Propriété 3.3 (Suite croissante d'interprétations et langages) *Soient V_T et V_N deux vocabulaires disjoints et V l'union de ces deux vocabulaires. Soit E un ensemble* totalement ordonné *d'interprétations de V_N sur V_T.*
Pour tout L langage sur V, $\bigcup\{\mathrm{I}(L) \mid \mathrm{I} \in E\} = (\bigcup E)(L)$

Preuve : En effet

$x \in \bigcup\{\mathrm{I}(L) \mid \mathrm{I} \in E\}$	
si et seulement si il existe $\mathrm{I} \in E$ telle que $x \in \mathrm{I}(L)$	par définition de $\bigcup$
si et seulement si il existe $\mathrm{I} \in E$, il existe $y \in L$ tels que $x \in \mathrm{I}(y)$	car $\mathrm{I}(L) = \bigcup\{\mathrm{I}(y) \mid y \in L\}$
si et seulement si il existe $y \in L$, il existe $\mathrm{I} \in E$ tels que $x \in \mathrm{I}(y)$	en permutant les "il existe"
si et seulement si il existe $y \in L$, $x \in \bigcup\{\mathrm{I}(y) \mid \mathrm{I} \in E\}$	par définition de $\bigcup$
si et seulement si il existe $y \in L$, $x \in (\bigcup E)(y)$	d'après la propriété 3.2
si et seulement si $x \in (\bigcup E)(L)$	car $(\bigcup E)(L) = \bigcup\{(\bigcup E)(y) \mid y \in L\}$

□

Propriété 3.4 *Soient X et Y deux ensembles et f une fonction de l'ensemble Y^X dans lui même. Soient $Z \subset Y^X$ et $x \in X$.*
Alors $\{(f(y))(x) \mid y \in Z\} = \{g(x) \mid g \in \{f(y) \mid y \in Z\}\}$

Preuve : En effet :

$u \in \{(f(y))(x) \mid y \in Z\}$	
si et seulement si il existe $y \in Z$ tel que $u = (f(y))(x)$	
si et seulement si il existe $y \in Z$ et il existe $g \in Y^X$ tels que $u = g(x)$ et $g = f(y)$	D'après les propriétés de l'égalité
si et seulement si il existe $g \in Y^X$ tel que $u = g(x)$ et il existe $y \in Z$ tel que $g = f(y)$	En permutant les quantificateurs
si et seulement si il existe $g \in Y^X$ tel que $u = g(x)$ et $g \in \{f(y) \mid y \in Z\}$	
si et seulement si il existe $u \in \{g(x) \mid g \in \{f(y) \mid y \in Z\}\}$	

□

Théorème 3.1 *La fonction f_S définie par un système d'équations S est continue.*

Preuve : Soit $S =< V_T, V_N, d >$ un système d'équations.
On montre que :

$$\boxed{\text{Pour tout } E \text{ ensemble } \textit{totalement ordonné} \text{ d'interprétations de } V_N \text{ sur } V_T,\ f_S(\bigcup E) = \bigcup\{f_S(\mathrm{I}) \mid \mathrm{I} \in E\}}$$

Puisqu'une suite croissante dénombrable est un cas particulier d'ensemble totalement ordonné, d'après la définition de la continuité 2.2, la propriété encadrée implique la continuité de f_S.
Pour montrer cette propriété, il suffit de vérifier que pour tout $A \in V_N$, $f_S(\bigcup E)(A) = (\bigcup\{(f_S(I)) \mid \mathrm{I} \in E\})(A)$.
En effet :

$$\begin{array}{rcll}
(f_S(\bigcup E))(A) & = & (\bigcup E)(\mathrm{d}(A)) & \text{D'après la définition 3.4 de } f_S \\
& = & \bigcup\{\mathrm{I}(\mathrm{d}(A)) \mid \mathrm{I} \in E\} & \text{D'après la propriété 3.3 appliquée au langage } \mathrm{d}(A) \\
& = & \bigcup\{(f_S(\mathrm{I}))(A) \mid \mathrm{I} \in E\} & \text{D'après la définition 3.4 de } f_S
\end{array}$$

Or, d'après la propriété 3.4, $\{(f_S(\mathrm{I}))(A) \mid \mathrm{I} \in E\} = \{\mathrm{J}(A) \mid \mathrm{J} \in \{f_S(\mathrm{I}) \mid \mathrm{I} \in E\}\}$.
On en déduit que $\bigcup\{(f_S(\mathrm{I}))(A) \mid \mathrm{I} \in E\} = \bigcup\{\mathrm{J}(A) \mid \mathrm{J} \in \{f_S(\mathrm{I}) \mid \mathrm{I} \in E\}\}$, d'où

$$\begin{array}{rcll} f_S(\bigcup E)(A) & = & \bigcup\{\mathrm{J}(A) \mid \mathrm{J} \in \{f_S(\mathrm{I}) \mid \mathrm{I} \in E\}\} & \\ & = & (\bigcup\{f_S(\mathrm{I}) \mid \mathrm{I} \in E\})(A) & \text{Car pour tout ensemble } F \text{ d'interprétations, } \bigcup\{\mathrm{J}(A) \mid J \in F\} = (\bigcup F)(A) \end{array}$$

□

Théorème 3.2 *Soit S un système d'équations sur l'ensemble V_T de terminaux et l'ensemble V_N de variables.*
Soit $\perp$ la plus petite interprétation de V_N sur V_T : pour tout $A \in V_N$, $\perp(A) = \emptyset$.
La plus petite solution de S est $\bigcup\{f_S^i(\perp) \mid i \in \mathbb{N}\}$.

Preuve : D'après le théorème ci-dessus 3.1, la fonction f_S est continue. La plus petite solution de S est le plus petit point fixe de f_S. D'après le théorème de Kleene, cette plus petite solution est $\bigcup\{f_S^i(\perp) \mid i \in \mathbb{N}\}$. □

Exemple 3.3

1. *Soit S le système de terminaux a, b, de variable A_0 et d'équation $A_0 = aA_0b + \varepsilon$. Alors :*
$$\begin{array}{lcl} f_S^1(\perp) & = & < A_0 \mapsto \{\varepsilon\} > \\ f_S^2(\perp) & = & < A_0 \mapsto \{\varepsilon, ab\} > \\ f_S^3(\perp) & = & < A_0 \mapsto \{\varepsilon, ab, aabb\} > \\ f_S^i(\perp) & = & < A_0 \mapsto \{a^jb^j \mid j < i\} \\ \bigcup\{f_S^i(\perp) \mid i \in \mathbb{N}\} & = & < A_0 \mapsto \{a^jb^j \mid j \in \mathbb{N}\} \end{array}$$
La dernière fonction est la plus petite solution (d'ailleurs unique) du système d'équations.
2. *Soit S, le système d'équations avec deux terminaux a, b, deux variables A_0, A_1 et les équations :*
$$\begin{array}{lcl} A_0 & = & aA_0 + bA_1 \\ A_1 & = & aA_1b + \varepsilon \end{array}$$
On a alors :
$$\begin{array}{lcl} f_S^1(\perp) & = & < A_0 \mapsto \emptyset, A_1 \mapsto \{\varepsilon\} > \\ f_S^2(\perp) & = & < A_0 \mapsto \{b\}, A_1 \mapsto \{\varepsilon, ab\} > \\ f_S^3(\perp) & = & < A_0 \mapsto \{ab, b, bab\}, A_1 \mapsto \{\varepsilon, ab, aabb\} > \\ f_S^4(\perp) & = & < A_0 \mapsto \{aab, ab, abab, b, bab, baabb\}, A_1 \mapsto \{\varepsilon, ab, aabb\} > \\ f_S^i(\perp) & = & < A_0 \mapsto \{a^jba^kb^k \mid j+k < i-1\}, A_1 \mapsto \{a^jb^j \mid j < i\} > \\ \bigcup\{f_S^i(\perp) \mid i \in \mathbb{N}\} & = & < A_0 \mapsto \{a^jba^kb^k \mid j, k \in \mathbb{N}\}, A_1 \mapsto \{a^jb^j \mid j \in \mathbb{N}\} > \end{array}$$
La dernière fonction est la plus petite solution (d'ailleurs unique) du système d'équations.

4 Équivalence entre langages algébriques et hors-contextes

Définition 4.1 *Un langage est algébrique si et seulement il est une composante de la plus petite solution d'un système d'équations. Plus formellement soit L un langage sur V_T. Ce langage est algébrique si et seulement si il y a un ensemble de variables V_N, disjoint de V_T, un système d'équations S sur V_T et V_N, de plus petite solution* I*, et une variable $A \in V_N$ telle que $L = \mathrm{I}(A)$.*

Le théorème suivant montre qu'un langage est algébrique si et seulement si il est hors-contexte.

Définition 4.2 *Soit $S =< V_T, V_N, d >$ un système d'équations. Soit $R = \{A \to \alpha \mid A \in V_N, \alpha \in \mathrm{d}(A)\}$ l'ensemble des règles associées au système S. Soit $A \in V_N$.*
La grammaire $G_{S,A} =< V_T, V_N, R, A >$ est la grammaire hors-contexte associée au système S et à la variable A.

En pratique le passage d'un système à un ensemble de règles hors-contextes consiste à remplacer le signe $=$ par la flêche $\to$ et le plus par la barre verticale. Par exemple le système
$$\begin{array}{lcl} A_0 & = & aA_0 + bA_1 \\ A_1 & = & aA_1b + \varepsilon \end{array}$$
est remplacé par les règles
$$\begin{array}{lcl} A_0 & \to & aA_0 | bA_1 \\ A_1 & \to & aA_1b | \varepsilon \end{array}$$

Théorème 4.1 *Soit $S =< V_T, V_N, d >$ un système d'équations.*
La plus petite solution du système est l'interprétation I *telle que pour tout $A \in V_N$, $\mathrm{I}(A) = L(G_{S,A})$.*
Autrement dit la composante de la variable A dans la plus petite solution du système est le langage engendré par la grammaire d'axiome A et dont les règles sont les règles associées au système.

Preuve : Soit $V = V_T \cup V_N$ et R l'ensemble des règles associées au système S.

1. L'interprétation I est une solution du système S.
   Par définition de I, il est clair que pour tout $\alpha \in V^*$, $\mathrm{I}(\alpha) = \{x \in V_T^* \mid \alpha \Rightarrow_R^* x\}$.

   (a) Pour tout $A \in V_N$, $\mathrm{I}(\mathrm{d}(A)) \subset \mathrm{I}(A)$.
   Soit $x \in \mathrm{I}(\mathrm{d}(A))$. Par définition de $\mathrm{I}(\mathrm{d}(A))$, il existe $\alpha \in \mathrm{d}(A)$ telle que $x \in \mathrm{I}(\alpha)$.
   Par définition de R, $A \to \alpha$ est une règle de la grammaire $G_{S,A}$.
   Puisque $x \in \mathrm{I}(\alpha)$, d'après la propriété de $\mathrm{I}(\alpha)$, $\alpha \Rightarrow_R^* x$
   Par suite x est un mot engendré par la grammaire $G_{S,A}$, donc $x \in \mathrm{I}(A)$.

   (b) Pour tout $A \in V_N$, $\mathrm{I}(A) \subset \mathrm{I}(\mathrm{d}(A))$.
   Soit $x \in \mathrm{I}(A)$. Par définition de I, le mot x est engendré par la grammaire $G_{S,A}$.
   Donc il existe α tel que $A \to \alpha$ est une règle de R et $\alpha \Rightarrow_R^* x$.
   Par définition de R, $\alpha \in \mathrm{d}(A)$. D'après la propriété de $\mathrm{I}(\alpha)$, $x \in \mathrm{I}(\alpha)$.
   Donc par définition de $\mathrm{I}(\mathrm{d}(A))$, $x \in \mathrm{I}(\mathrm{d}(A))$.

2. L'interprétation I est la plus petite solution du système S.
   Soit J une autre solution du système. Soit $A \in V_N$, $x \in V_T^*$, n un entier.
   Par induction sur n, on montre que pour tout n, si $A \Rightarrow_R^n x$ alors $x \in \mathrm{J}(A)$.
   Suppposons que pour tout p où $p < n$, si $A \Rightarrow_R^p x$ alors $x \in \mathrm{J}(A)$: c'est l'hypothèse d'induction.

   (a) Soit $n = 0$. Montrons que si $A \Rightarrow_R^n x$ alors $x \in \mathrm{J}(A)$.
   Pour $n = 0$, on aurait $A = x$ ce qui est impossible car A est une variable et x une chaîne terminale. Donc l'implication est vraie.

   (b) Soit $n > 0$. Supposons $A \Rightarrow_R^n x$.
   Il y a une règle $A \to \alpha$ dans R et $\alpha \Rightarrow_R^{n-1} x$.
   Soit $B_{(i|i \text{ de } 1 \text{ à } p)}$ la suite des symboles de α.
   D'après la propriété fondamentale 1.1 des grammaires hors-contextes :
   Pour i de 1 à p, $B_i \Rightarrow_R^{k_i} x_i$ où $n - 1 = \Sigma_i^p k_i$ et $x = x_1 \ldots x_p$.
   Quand $B_i \in V_T$, par définition des dérivations on a $B_i = x_i$ et par définition des interprétations, $\mathrm{J}(B_i) = \{B_i\}$, donc $x_i \in \mathrm{J}(B_i)$.
   Quand $B_i \in V_N$, puisque $k_i < n$, l'hypothèse d'induction est applicable, on a de même $x_i \in \mathrm{J}(B_i)$.
   Par suite $x_1 \ldots x_p \in \mathrm{J}(B_1 \ldots B_p)$, c'est-à-dire $x \in \mathrm{J}(\alpha)$.
   Puisque $\alpha \in \mathrm{d}(A)$, il résulte que $x \in \mathrm{J}(\mathrm{d}(A))$ et puisque J est une solution du système S, $\mathrm{J}(A) = \mathrm{J}(\mathrm{d}(A))$, donc $x \in \mathrm{J}(A)$.

   Par le principe d'induction, pour tout n, si $A \Rightarrow_R^n x$ alors $x \in \mathrm{J}(A)$, donc $\mathrm{I} \subset \mathrm{J}$.
   Par suite I est bien la plus petite solution du système d'équations.

□

5 Transformation de systèmes d'équations

Définition 5.1 (Relation entre systèmes d'équations) *Soit V_T un ensemble de terminaux et V_N un ensemble de variables. Soit $S =< V_T, V_N, d >$ et $S' =< V_T, V_N, d' >$ deux systèmes d'équations.*
S' est conséquence *de S, si toute solution de S est solution de S'.*
Ces deux systèmes sont équivalents *s'ils sont les mêmes solutions.*

5.1 Substitution simple

Définition 5.2 (Substitution simple) *Le système S' est obtenu par subsitution simple sur le système S s'il est obtenu en remplaçant dans une définition de S,* une *occurrence d'une variable par sa définition.*
Plus formellement soient $S =< V_T, V_N, d >$ et $S' =< V_T, V_N, d' >$ deux systèmes. S' est obtenu par substitution simple sur S, s'il existe $A, B \in V_N$, $\alpha B \beta \in \mathrm{d}(A)$ [1] *tel que :*
d' *est identique à* d *sauf pour A et* $\mathrm{d}'(A) = (\mathrm{d}(A) - \{\alpha B \beta\}) \cup \alpha \mathrm{d}(B) \beta$.

On va montrer que cette substitution change un système en un autre équivalent. On a besoin dans cette preuve d'une propriété sur les langages, que nous montrons ci-dessous.

Propriété 5.1 *Soient L, M, N, P, Q, cinq langages sur le même vocabulaire vérifiant les égalités*

$$\begin{array}{lcl} P & = & L \cup MQN \\ Q & = & L \cup MPN \end{array}$$

Il en résulte que $P = Q$

Preuve :

1. Supposons $\varepsilon \in M$ et $\varepsilon \in N$.
   Puisque $\varepsilon \in M$ et $\varepsilon \in N$, on a $Q \subset MQN$. D'après la première égalité $MQN \subset P$. Donc $Q \subset P$.
   De façon analogue, grâce à la deuxième égalité, on a $P \subset Q$, donc $P = Q$.
2. Supposons $\varepsilon \notin M$ ou $\varepsilon \notin N$.
   On montre par induction sur x que $x \in P$ si et seulement si $x \in Q$.
   (a) Supposons $x = \varepsilon$.
   Supposons $x \in P$. Puisque tout mot de MQN n'est pas vide, d'après la première égalité $x \in L$ et d'après la deuxième $x \in Q$.
   Réciproquement, de façon analogue si $x \in Q$ alors $x \in P$.
   (b) Supposons $x \neq \varepsilon$.
   On admet l'hypothèse d'induction que tout mot, de longueur inférieure à celle de x, est élément de P si et seulement si il est élément de Q.
   i. Supposons $x \in L$. Des deux égalités, il résulte immédiatement que $x \in P$ si et seulement si $x \in Q$.
   ii. Supposons $x \notin L$.
   Supposons $x \in P$.
   D'après la première équation $x \in MQN$, donc $x = uvw$ avec $u \in M$, $v \in Q, w \in N$. Puisque $\varepsilon \notin M$ ou $\varepsilon \notin N$, la longueur de v est inférieure à celle de x, donc par hypothèse d'induction $v \in P$.
   Par suite $x = uvw \in MPQ$, donc d'après la deuxième égalité $x \in Q$.
   De façon analogue, $x \in Q$ implique $x \in P$
   Donc $x \in P$ si et seulement si $x \in Q$. Par le principe d'inducion, c'est vrai pour tout mot x, donc $P = Q$.

□

Théorème 5.1 *Soient deux systèmes d'équations dont l'un est obtenu par une substitution simple appliquée à l'autre. Ces deux systèmes sont équivalents.*

Preuve : Soient $S =< V_T, V_N, d >$ et $S' =< V_T, V_N, d' >$ deux systèmes. S' est obtenu par substitution simple sur S. Donc il existe $A, B \in V_N$, $\alpha B \beta \in \mathrm{d}(A)$ tel que :
d' est identique à d sauf pour A et $\mathrm{d}'(A) = (\mathrm{d}(A) - \{\alpha B \beta\}) \cup \alpha \mathrm{d}(B) \beta$.

1. Soit I solution de S. Montrons que c'est également une solution de S'.
   Il suffit de vérifier que $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}'(A))$.

$$\begin{array}{lcll} \mathrm{I}(A) & = & \mathrm{I}(\mathrm{d}(A)) & \text{Car I est solution de } S \\ & = & \mathrm{I}(\mathrm{d}(A) - \{\alpha B \beta\}) \cup \mathrm{I}(\alpha B \beta) & \text{Car } \mathrm{d}(A) = (\mathrm{d}(A) - \{\alpha B \beta\}) \cup \{\alpha B \beta\} \\ & = & \mathrm{I}(\mathrm{d}(A) - \{\alpha B \beta\}) \cup \mathrm{I}(\alpha \mathrm{d}(B) \beta) & \text{Car } \mathrm{I}(B) = \mathrm{I}(\mathrm{d}(B)) \text{ puisque I est solution de } S \\ & = & \mathrm{I}(\mathrm{d}'(A)) & \text{Par définition de } \mathrm{d}'(A) \end{array}$$

[1] Rappelons que $\mathrm{d}(A)$ est la définition de A

2. Soit I une solution de S'. Montrons que c'est également une solution de S.
   Il suffit de vérifier que $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A))$.
   Puisque I est solution de S', $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}'(A)) = \mathrm{I}(\mathrm{d}(A) - \{\alpha B \beta\}) \cup \mathrm{I}(\alpha \mathrm{d}(B) \beta)$.
   (a) Supposons $A \neq B$.
   Puisque seule l'équation définissant A est modifiée, $\mathrm{d}(B) = \mathrm{d}'(B)$.
   Puisque I est solution de S', $\mathrm{I}(B) = \mathrm{I}(\mathrm{d}'(B)) = \mathrm{I}(\mathrm{d}(B))$.
   Donc en remplaçant $\mathrm{I}(\mathrm{d}(B))$ par la valeur qui lui est égale $\mathrm{I}(B)$ dans la valeur de $\mathrm{I}(A)$, on obtient :
   $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A) - \{\alpha B \beta\}) \cup \mathrm{I}(\alpha B \beta) = \mathrm{I}(\mathrm{d}(A))$.
   Donc I est solution de S.
   (b) Supposons $A = B$.
   On a alors :
   $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A) - \{\alpha A \beta\}) \cup \mathrm{I}(\alpha)\mathrm{I}(\mathrm{d}(A))\mathrm{I}(\beta)$ Car I est solution de S' et $\mathrm{d}'(A) = (\mathrm{d}(A) - \{\alpha A \beta\}) \cup \alpha \mathrm{d}(A) \beta$
   $\mathrm{I}(\mathrm{d}(A)) = \mathrm{I}(\mathrm{d}(A) - \{\alpha A \beta\}) \cup \mathrm{I}(\alpha)\mathrm{I}(A)\mathrm{I}(\beta)$ Car $\mathrm{d}(A) = (\mathrm{d}(A) - \{\alpha A \beta\}) \cup \{\alpha A \beta\}$
   D'après la proposition 5.1 en instanciant L, M, N, P, Q respectivement par les ensembles $I(\mathrm{d}(A) - \{\alpha A \beta\})$, $I(\alpha)$, $I(\beta)$, $\mathrm{I}(A)$ et $\mathrm{I}(\mathrm{d}(A))$, on obtient $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A))$
   Donc I est solution de S.

□

5.2 Substitution générale

Définition 5.3 (Substitution générale) *Le système S' est obtenu par substitution générale sur le système S s'il est obtenu en remplaçant dans les définitions de S, des occurrences de variables par leur définition.*
Plus formellement soient $S =< V_T, V_N, \mathrm{d} >$ et $S' =< V_T, V_N, \mathrm{d}' >$ deux systèmes, S' est obtenu par substitution générale sur le système S si :
Pour tout $A \in V_N$ $\mathrm{d}'(A) = \bigcup\{E_\alpha \mid \alpha \in \mathrm{d}(A)\}$ *et*
tout E_α est de la forme $\beta_0 \mathrm{d}(B_1)\beta_1 \ldots \mathrm{d}(B_n)\beta_n$ où $n \geq 0$, $B_1, \ldots B_n \in V_N$ et $\alpha = \beta_0 B_1 \beta_1 \ldots B_n \beta_n$.

Notons qu'une subsitution simple est un cas particulier d'une substitution simple.
La substitution générale, contrairement à la substitution simple, ne préserve pas nécessairement l'ensemble des solution d'un système d'équations. Cependant, comme on le prouve ci-dessous, elle préserve la plus petite solution.
Considérons l'exemple suivant. Soit le système S de terminaux a, b et de variables A, B et d'équations :

$$\begin{array}{lcl} A & = & B + a \\ B & = & A + b \end{array}$$

En remplaçant à droite des équations, A et B par leur définition, on obtient le système S' suivant :

$$\begin{array}{lcl} A & = & A + a + b \\ B & = & B + a + b \end{array}$$

L'interprétation $A = a + b + \varepsilon, B = a + b$ est solution de S' mais n'est pas une solution de S. Donc la substitution générale ne préserve pas toutes les solutions.
Par contre $A = a + b, B = a + b$ est la plus petite solution des deux systèmes.

Propriété 5.2 *Soit S' un système obtenu par substitution générale sur le système S.*
Toute solution de S est solution de S'.

Preuve : Nous reprenons, pour les deux systèmes, les notations de la définition 5.3.
Soit $E_\alpha = \beta_0 \mathrm{d}(B_1)\beta_1 \ldots \mathrm{d}(B_n)\beta_n$ où $n \geq 0$, $B_1, \ldots B_n \in V_N$ et $\alpha = \beta_0 B_1 \beta_1 \ldots B_n \beta_n$.
Si I est une solution de S, alors pour tout i de 1 à n, $\mathrm{I}(B_i) = \mathrm{I}(\mathrm{d}(B_i))$ et par suite $\mathrm{I}(E_\alpha) = \mathrm{I}(\alpha)$.
Donc pour tout $A \in V_N$, par définition de $\mathrm{d}'(A)$, $\mathrm{I}(\mathrm{d}(A)) = \mathrm{I}(\mathrm{d}'(A))$.
Puisque I est solution de S, pour tout $A \in V_N$, $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}(A))$, donc aussi $\mathrm{I}(A) = \mathrm{I}(\mathrm{d}'(A))$.
Par suite I est aussi solution de S'. □

Propriété 5.3 *Soit S' un système obtenu par substitution générale sur le système S.*
La plus petite solution de S est inférieure ou égale à la plus petite solution de S' pour l'ordre sur les interprétations défini comme comme l'extension de l'ordre de l'inclusion entre ensembles.

Preuve : Nous reprenons, pour les deux systèmes, les notations de la définition 5.3.
Soient respectivement I et I′ les plus petites solutions de S et S'.
Rappelons que f_S et $f_{S'}$ sont les fonctions définies repectivement par S et S'.
D'après le théorème 3.1 ces deux fonctions sont continues, donc d'après la preuve du théorème de Kleene les suites $f_S^i(\perp)$ et $f_{S'}^i(\perp)$ sont croissantes, et d'après le théorème 3.2, $\mathrm{I} = \bigcup\{f_S^i(\perp) \mid i \in \mathbb{N}\}$, et $\mathrm{I}' = \bigcup\{f_{S'}^i(\perp) \mid i \in \mathbb{N}\}$.

1. Soit I une interprétation telle que $\mathrm{I} \subset f_S(\mathrm{I})$. Montrons que $f_S(\mathrm{I}) \subset f_{S'}(\mathrm{I})$
   Puisque $\mathrm{I} \subset f_S(\mathrm{I})$, par définition de f_S, pour tout $A \in V_N$, $\mathrm{I}(A) \subset \mathrm{I}(\mathrm{d}(A))$.
   Soit A une variable et x une chaîne terminale telle que $x \in (f_S(\mathrm{I})(A)$. Il suffit de montrer que $x \in (f_{S'}(\mathrm{I}))(A)$.
   Puisque $x \in (f_S(\mathrm{I}))(A) = \mathrm{I}(\mathrm{d}(A))$, il existe $\alpha \in \mathrm{d}(A)$ tel que $x \in \mathrm{I}(\alpha)$.
   Puisque S' est obtenue par substitution générale, on a :
   $\alpha = \beta_0 B_1 \beta_1 \ldots B_n \beta_n$ où pour i de 1 à n, B_i est une variable et $\beta_0 \mathrm{d}(B_1)\beta_1 \ldots \mathrm{d}(B_n)\beta_n \subset \mathrm{d}'(A)$.
   Puisque pour tout $A \in V_N$, $\mathrm{I}(A) \subset \mathrm{I}(\mathrm{d}(A))$, il en résulte que pour i de 1 à n, $\mathrm{I}(B_i) \subset \mathrm{I}(\mathrm{d}(B_i))$.
   Puisque la concaténation de langages est croissante, on en déduit que $\mathrm{I}(\alpha) \subset \mathrm{I}(\beta_0 \mathrm{d}(B_1)\beta_1 \ldots \mathrm{d}(B_n)\beta_n)$.
   Puisque $\beta_0 \mathrm{d}(B_1)\beta_1 \ldots \mathrm{d}(B_n)\beta_n \subset \mathrm{d}'(A)$, nous concluons que $\mathrm{I}(\alpha) \subset \mathrm{I}(\mathrm{d}'(A))$.
   Comme $x \in \mathrm{I}(\alpha)$, on a $x \in \mathrm{I}(\mathrm{d}'(A))$; de plus, par définition de $f_{S'}$, $\mathrm{I}(\mathrm{d}'(A)) = (f_{S'}(\mathrm{I}))(A)$, donc $x \in (f_{S'}(\mathrm{I}))(A)$.
2. Montrons par induction sur i que pour tout $i \in \mathbb{N}$, $f_S^i(\perp) \subset f_{S'}^i(\perp)$.
   Pour $i = 0$ c'est trivial car $f_S^0(\perp) = \perp = f_{S'}^0(\perp)$
   Supposons que $f_S^i(\perp) \subset f_{S'}^i(\perp)$.
   On a rappelé ci-dessus que $f_S^i(\perp)$ est une suite croissante, donc $f_S^i(\perp) \subset f_S(f_S^i(\perp))$.
   Par suite on peut appliquer le résultat 1 du premier cas à l'interprétation $f_S^i(\perp)$, donc :
   (*) $f_S(f_S^i(\perp)) \subset f_{S'}(f_S^i(\perp))$.
   Puisque toute fonction continue est croissante d'après la propriété 2.1, $f_{S'}$ est croissante. Comme par hypothèse d'induction $f_S^i(\perp) \subset f_{S'}^i(\perp)$, il resulte de la croissance de $f_{S'}$ que :
   (**) $f_{S'}(f_S^i(\perp)) \subset f_{S'}(f_{S'}^i(\perp))$.
   Donc d'après **(*)** et **(**)**, on a $f_S^{i+1}(\perp) \subset f_{S'}^{i+1}(\perp)$

Puisque pour tout $i \in \mathbb{N}$, $f_S^i(\perp) \subset f_{S'}^i(\perp)$, que I est le plus petit majorant de la suite $f_S^i(\perp)$ et I' le plus petit majorant de la suite $f_{S'}^i(\perp)$, il en résulte que $I \subset I'$. □

Théorème 5.2 *Soit S' un système obtenu par substitution générale sur le système S.*
Les plus petites solutions de S et de S' sont identiques.

Preuve : Soient respectivement I et I′ les plus petites solutions de S et S'.
D'après la propriété 5.2, I est aussi solution de S' donc $\mathrm{I}' \subset \mathrm{I}$.
D'après la propriété 5.3, $\mathrm{I} \subset \mathrm{I}'$. Donc $\mathrm{I} = \mathrm{I}'$. □

Chapitre VIII

Transformations des grammaires hors-contexte

1 Rappels du chapitre précédent

Dans ce chapitre on présente des transformations de grammaires, préservant les langages engendrés et utiles pour rédiger des programmes efficaces d'analyse, c'est-à-dire des programmes pour reconnaître si un mot est engendré par une grammaire.
La justification de certaines de ces transformations repose sur des résultats du chapitre précedent qui seront brièvement rappelés sous une forme qui devrait éviter la lecture de ce chapitre précédent à ceux qui souhaitent dans un premier temps faire l'impasse sur ces preuves.

1.1 Grammaires et équations

Une grammaire hors-contexte $G =< V_T, V_N, S, R >$ peut être vue comme un système d'équations, avec une équation par non-terminal. Soit $V = V_T \cup V_N$.
Soit A un non-terminal (que l'on appelle aussi une variable) et soit $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$ où $n > 0$ et $\alpha_1, \alpha_2, \ldots \alpha_n \in V^*$ l'ensemble des règles d'entête A. Cet ensemble de règles est vu comme une équation définissant A que l'on l'écrit $A = \alpha_1 + \alpha_2 + \cdots + \alpha_n$.
Un système d'équations est la donnée d'un triplet $< V_T, V_N, \mathrm{d} >$, où V_T est l'ensemble des terminaux, V_N est l'ensemble des variables, d est l'ensemble des équations entre une variable et une somme de mots dont les lettres sont des terminaux ou des variables.
Les *valeurs des variables* sont des langages sur le vocabulaire terminal V_T.
Une *interprétation du système* est une suite d'égalités entre les variables du système et des valeurs.
Une interprétation est une *solution* du système, si les valeurs obtenues en remplaçant chaque variable par sa valeur dans les deux membres de chaque équation du système, sont égales.
On ordonne ainsi les interprétations : soient I et J deux interprétations d'un système, I est inférieure ou égale à J (ce qu'on note I $\subset$ J) si et seulement si pour toute variable A du système la valeur de A donnée par I est un sous-ensemble de la valeur de A donnée par J.

Exemple 1.1 *Soit la grammaire G de vocabulaire terminal a, b, de vocabulaire non-terminal A, B, d'axiome A et de règles :*

$$\begin{array}{lcl} A & \rightarrow & aA|B \\ B & \rightarrow & \varepsilon|aBb \end{array}$$

On peut voir G comme un système d'équations :

$$\begin{array}{lcl} A & = & aA + B \\ B & = & \varepsilon + aAb \end{array}$$

L'interprétation $A = \{ab, a, aba\}, B = \{b^n a \mid n \in \mathbb{N}\}$ n'est pas une solution du système.

Par contre l'interprétation $A = \{a^n b^p \mid n \geq p\}, B = \{a^n b^n \mid n \geq 0\}$ est une solution de ce système d'équations, qui n'a d'ailleurs qu'une seule solution.

Il y a des systèmes d'équations qui ont plus qu'une solution. Par exemple le système $A = A + a + b$ de variable A et de terminaux a, b possède une plus petite solution $A = \{a, b\}$. Mais $A = L$ où L est un langage quelconque tel que $a, b \in L$ est une aussi une solution.
La relation entre les solutions et les grammaires est résumée par le résultat suivant.

Théorème 1.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. Une interprétation est* la plus petite solution *du système associé à la grammaire si et seulement si pour tout non-terminal A, la valeur de A dans l'interprétation est le langage engendré par le non-terminal A avec les règles de G.*

Pour les courageux, qui veulent lire la preuve, voir le théorème 4.1 du chapitre IX.

1.2 Des substitutions, qui préservent les langages

Définition 1.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte et $V = V_T \cup V_N$. On appelle définition du non-terminal A et l'on note par $\mathrm{d}(A)$ l'ensemble des parties droites des règles d'entête A.*
La grammaire $G' =< V_T, V_N, S, R' >$ est obtenue par substitution générale sur G, si les règles de G' sont obtenues en remplaçant dans les parties droites de règles des variables par leurs définitions.
Plus précisément pour chaque règle $A \to \alpha$ de la grammaire G, il existe $n \geq 0$, des mots $\beta_0, \beta_1, \ldots \beta_n$ de V^*, des variables $B_1, \ldots B_n \in V_N$ tel que
- $\alpha = \beta_0 B_1 \beta_1 \ldots B_n \beta_n$
- *$d'(A)$ la définition de A dans R' est l'ensemble $\beta_0 \mathrm{d}(B_1) \beta_1 \ldots \mathrm{d}(B_n) \beta_n$*

Théorème 1.2 *Soit G' obtenue par substitution générale sur la grammaire G. Les deux grammaires sont équivalentes.*

Pour les courageux, qui veulent lire la preuve, voir le théorème 5.2 du chapitre IX.

1.3 Treillis fini et calcul de points fixes

On rappelle qu'un treillis est un ensemble ordonné dans lequel tout sous-ensemble fini non vide a une plus petite borne supérieure et une plus petite borne inférieure.

Théorème 1.3 *Soit E un treillis* fini *ayant un plus petit élément $\perp$. Soit n le nombre d'éléments de E.*
Soit $f : E \mapsto E$ une fonction croissante.
La suite $f^i(\perp)$ est une suite croissante.
Le plus petit point fixe de f est $f^j(\perp)$ où j est le plus petit entier tel que $j < n$ et $f^j(\perp) = f^{j+1}(\perp)$.

Pour les courageux, qui veulent lire la preuve, voir le paragraphe 2.3.

2 Réduction d'une grammaire hors-contexte

Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. Soit $V = V_T \cup V_N$.

2.1 Calcul de l'ensemble des symboles productifs

Le non-terminal $A \in V_N$ est productif si et seulement si il existe une chaîne terminale x telle que $A \Rightarrow^* x$. Un non-terminal improductif ne figure dans aucune dérivation dont l'extrémité est une chaîne terminale.
L'ensemble des symboles productifs peut aussi être défini par induction comme étant le plus petit ensemble de non-terminaux tel que A est productif si et seulement si il y a une règle $A \to \alpha$ dans la grammaire, telle que toutes les variables de α sont productives. Prouvons que ces deux manières de définir l'ensemble des productifs sont équivalentes.

Théorème 2.1 *Soit $f : \mathcal{P}(V_N) \mapsto \mathcal{P}(V_N)$ la fonction suivante :*
Pour tout $E \subset V_N$, $f(E) = \{A \in V_N \mid \exists \alpha \in (V_T \cup E)^* \text{ tel que } A \to \alpha \in R\}$.
L'ensemble des symboles productifs est le plus petit point fixe de f

Preuve : Soit P l'ensemble des symboles productifs.

1. L'ensemble P des symboles productifs vérifie $f(P) = P$.
   (a) Soit $A \in f(P)$. Donc il existe une règle $A \to \alpha$ dont tous les symboles de α sont des terminaux ou produisent des chaînes terminales. Par composition des dérivations A produit une chaîne terminale, donc $A \in P$.
   (b) Soit $A \in P$. Il existe α et une chaîne terminale x tel que $A \to \alpha \in R$ et $\alpha \Rightarrow^* x$. Par décomposition des dérivations, chaque symbole de α est un terminal ou un non-terminal produisant une chaîne terminale. Donc $\alpha \in (V_T \cup P)^*$ et par suite $A \in f(P)$.
2. P est la plus petite solution.
   Soit Q une solution quelconque. Il suffit de vérifier que $P \subset Q$.
   Montrons par induction sur i que :
   (*) pour tout non-terminal A et toute chaîne terminale x tel que $A \Rightarrow^i x$, on a $A \in Q$.
   Pour $i = 0$, la propriété est vérifiée car $A \Rightarrow^i x$ est impossible. Soit $i > 0$. Supposons, par induction, que pour tout $j < i$, pour tout non-terminal A et toute chaîne terminale x tel que $A \Rightarrow^j x$ on a $A \in Q$.
   Soit le non-terminal A et la chaîne terminale x tels que $A \Rightarrow^i x$.
   Puisque $i > 0$, il existe α tel que $A \to \alpha \in R$ et $\alpha \Rightarrow^{i-1} x$.
   Par décomposition des dérivations, chaque non-terminal de α engendre en au plus $i-1$ pas une chaîne terminale, donc par hypothèse d'induction est élément de Q.
   Par suite $\alpha \in (V_T \cup Q)^*$, donc $A \in f(Q)$. Puique $Q = f(Q)$, on a $A \in Q$.
   Par application du principe d'induction, la propriété **(*)** est vraie pour tout entier i.
   Soit $A \in P$. Il existe i et une chaîne terminale x telle que $A \Rightarrow^i x$.
   D'après la propriété **(*)**, vraie pour tout i, on a $A \in Q$

□

Le théorème ci-dessus nous donne un moyen de calculer l'ensemble P des non-terminaux productifs. En effet f est une fonction croissante sur le treillis fini $\mathcal{P}(V_N)$, ordonné par l'inclusion et dont le plus petit élément est $\emptyset$. Puisque P est le plus petit point fixe de f, d'après le théorème 1.3, $P = f^j(\emptyset)$ où j est le plus petit entier tel que $f^j(\emptyset) = f^{j+1}(\emptyset)$. D'après ce même théorème, $f^i(\emptyset)$ est une suite croissante d'ensembles. Soit n le nombre de non-terminaux, puisque $f^i(\emptyset)$ est une suite croissante, on a $j < n$.
Donc la valeur de P est obtenue en appliquant au plus n fois la fonction f.

Exemple 2.1 *Soit la grammaire ayant un terminal a, les non-terminaux A, B, C, D, E et de règles $A \to B$, $B \to C$, $C \to D$, $D \to a$, $D \to E$, $E \to EA$.*
On calcule la suite $f^i(\emptyset)$ jusqu'à la stabilité de la suite :

$$\begin{array}{lcl} f^0(\emptyset) & = & \emptyset \\ f^1(\emptyset) & = & \{D\} \\ f^2(\emptyset) & = & \{D,C\} \\ f^3(\emptyset) & = & \{D,C,B\} \\ f^4(\emptyset) & = & \{D,C,B,A\} \\ f^5(\emptyset) & = & \{D,C,B,A\} \end{array}$$

Donc le non-terminal E est improductif et les autres terminaux sont productifs.

Exercice 2.1 *Montrer que $f^i(\emptyset)$ est l'ensemble des symboles produisant une chaîne terminale avec un arbre de dérivation de hauteur au plus égale à i.*

2.2 Calcul de l'ensemble de l'ensemble des symboles accessibles

Un non-terminal B est *accessible* à partir du non-terminal A si et seulement si il existe deux chaînes $\alpha, \beta \in V^*$ telles que $A \Rightarrow^* \alpha B \beta$.

Un non-terminal est accessible relativement à une grammaire si et seulement si il est accessible à partir de l'axiome de la grammaire.
Évidemment un non-terminal inacessible ne figure dans aucune dérivation à partir de l'axiome, donc est inutile.
Soit d la relation de dépendance sur V_N définie par :

AdB si et seulement il existe deux chaînes $\alpha, \beta \in V^*$ telles que $A \rightarrow \alpha B \beta \in R$.
Notons d^* la fermeture reflexive et transitive de la relation d. Il est évident que B est *accessible* à partir du non-terminal A si et seulement si $A\ d^* B$.
Comme la relation d est une relation sur l'ensemble fini des non-terminaux, il y a des algorithmes pour la calculer.
Nous donnons ci-dessus l'algorithme de Roy-Warshall, qui est applicable à *tout* ensemble fini.
Soit A_i où i va de 1 à n, une liste sans répétition des éléments de V_N.
Notons d_k la relation suivante : $A_i d_k A_j$ si et seulement il y a un chemin (pour la relation d) entre A_i et A_j dont les sommets *intermédiaires* sont d'indice au plus égal à k.
Il est (presque) évident que ces relations vérifient :

1. $A_i d_0 A_j$ si et seulement si $A_i d A_j$ ou $i = j$
2. $d^* = d_n$ car n est le plus grand indice possible
3. Soit $k > 0$. $A_i d_k A_j$ si et seulement si $A_i d_{k-1} A_j$ ou ($A_i d_{k-1} A_k$ et $A_k d_{k-1} A_j$) car s'il y a un chemin entre A_i et A_j passant par des sommets d'indice au plus k, il y un chemin entre A_i et A_j passant une seule fois par un tel sommet.

Cela donne un calcul de d^* en un temps de l'ordre de n^3, car il suffit de calculer chacune des n relations d_k, où $k > 0$, et que le calcul de d_k necessite de calculer les n^2 couples de cette relation.

2.3 Construction d'une grammaire réduite

Définition 2.1 *Une grammaire hors-contexte est réduite si et seulement si tous ses symboles sont productifs et accessibles.*

Théorème 2.2 *Soit G une grammaire hors-contexte telle que $L(G) \neq \emptyset$. On peut construire une grammaire réduite équivalente à G.*

Preuve : Soit $G =< V_T, V_N, S, R >$.

1. Élimination des improductifs.
   Soit P l'ensemble des symboles productifs de la grammaire.
   Puisque $L(G)$ n'est pas vide, P comporte au moins l'axiome S.
   Soit R' l'ensemble des règles de G ne comportant pas de symboles improductifs et soit $G' =< V_T, P, S, R' >$.
   On montre aisément que pour tout $A \in P$ et toute chaîne terminale x : $A \Rightarrow_G^* x$ si et seulement si $A \Rightarrow_{G'}^* x$.
   Puisque S est l'axiome de G', il en résulte que les deux grammaires sont équivalentes.
   Puisque P est l'ensemble des non-terminaux de G', il en résulte que tous les non-terminaux de G' sont productifs.
2. Élimination des inaccessibles.
   Soit Q l'ensemble des symboles accessibles de la grammaire G'.
   Soit R'' l'ensemble des règles de G' ne comportant pas de symboles inaccessibles et soit $G'' =< V_T, Q, S, R'' >$.
   On montre aisément les propriétés suivantes de G''
   (a) pour tout $\alpha \in V^*$, si $S \Rightarrow_{G'}^* \alpha$ alors $S \Rightarrow_{G''}^* \alpha$.
   La preuve est une preuve facile mais fastidieuse sur la longueur des dérivations.
   (b) pour tout $\alpha \in V^*$, si $S \Rightarrow_{G''}^* \alpha$ alors $S \Rightarrow_{G'}^* \alpha$.
   Ce résultat résulte de ce que toute règle de G'' est une règle de G'.
   (c) pour tout $A \in Q$ et toute chaîne terminale x, si $A \Rightarrow_{G'} x$ alors $A \Rightarrow_{G''} x$.
   La preuve est une preuve facile mais fastidieuse sur la longueur des dérivations.

D'après la propriété 2a, tous les symboles accessibles dans G', le sont dans G'', donc tous les non-terminaux de G'' sont accessibles.
D'après la propriété 2c et le fait que tous les non-terminaux de G' sont productifs, il résulte que tous les non-terminaux de G'' sont productifs.
Donc la grammaire G'' est réduite.
Des propriétés 2a et 2b, il résulte que les grammaires G' et G'' sont équivalentes.

Puisque les grammaires G et G' sont équivalentes, il en résulte que les grammaires G et G'' sont équivalentes.
Puisque l'on a vu au paragraphe 2.1 comment calculer P, l'ensemble des symboles productifs de G, et que l'on a vu au paragraphe 2.2 comment calculer Q, l'ensemble des symboles accessibles de G', on sait comment construire une grammaire G'' réduite et équivalente à G. □

Remarque 2.1 *On peut résumer ainsi la transformation d'une grammaire en une grammaire réduite équivalente, enlever d'abord les non-terminaux improductifs (et les règles qui en comportent) puis enlever les non-terminaux inaccessibles (et les règles qui en comportent). Si on procéde dans l'ordre inverse, il n'est pas certain que la grammaire obtenue est réduite comme le montre l'exemple suivant.*
Soit G la grammaire de terminaux a,b,c, de non-terminaux S,A,C,D,E, d'axiome S et de règles $S \to A$,$S \to a$, $A \to CD$, $C \to c$, $D \to A$, $E \to b$.
Le non-terminal E est le seul symbole inacessible. En l'enlevant, on obtient la grammaire G' de non-terminaux S,A,C,D, d'axiome S et de règles $S \to A$,$S \to a$, $A \to CD$, $C \to c$, $D \to A$.
Les non-terminaux A,D, sont les seuls symboles improductifs. En les enlevant, on obtient la grammaire G'' de non-terminaux S,C, d'axiome S et de règles $S \to a$,$C \to c$.
La grammaire obtenue comporte le symbole inaccessible C : elle n'est pas réduite.

3 Élimination des ε-règles

Savoir éliminer les ε-règles n'a de nos jours aucun intérêt pratique, car les analyseurs savent les utiliser. Mais cela permet de justifier ce qui avait été dit sans preuve au chapitre II : tout langage hors-contexte ne comportant pas la chaîne vide, est sous-contexte. En effet les règles hors-contextes, dont la partie droite n'est pas vide, sont aussi des règles sous-contextes, comme on le voit dans la classification des grammaires 3.3 du chapitre II.

Théorème 3.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. On peut construire une grammaire hors-contexte $G' =< V_T, V_N, S, R' >$ sans ε-règle telle que $L(G') = L(G) - \{\varepsilon\}$.*

Preuve : On montre d'abord comment construire l'ensemble des non-terminaux engendrant la chaîne vide, puis on détaille la construction de G', enfin on montre que $L(G') \subset L(G) - \{\varepsilon\}$ et $L(G) - \{\varepsilon\} \subset L(G')$.

1. Construction de E l'ensemble des non-terminaux engendrant la chaîne vide.
   Cette construction est analogue à celle de l'ensemble de symboles productifs étudiée au paragraphe 2.1, aussi nous laissons au lecteur les justifications plus précises de cette construction.
   Soit $f : \mathcal{P}(V_N) \mapsto \mathcal{P}(V_N)$ la fonction suivante :
   Pour tout $X \subset V_N$, $f(X) = \{A \in V_N \mid \exists \alpha$ tel que $A \to \alpha$ et $\alpha \in X^*\}$.
   Il est clair que f est croissante et que E est le plus petit point fixe de f.
   Donc $E = f^j(\emptyset)$ où j et le plus petit indice tel que $f^j(\emptyset) = f^{j+1}(\emptyset)$.
   De plus, puisque la suite $f^i(\emptyset)$ est croissante, j est inférieur au nombre de non-terminaux de la grammaire.

2. Construction de R'.
   R' est obtenu en effaçant,dans les parties droites des règles de G, de *toutes* les manières possibles les non-terminaux produisant la chaîne vide.
   Plus formellement R' est l'ensemble de règles $A \to B_1 \dots B_p$ où
   – $p > 0$ et pour i de 1 à p, $B_i \in V$
   – il existe une règle de G s'écrivant $A \to \alpha_1 B_1 \dots \alpha_p B_p \alpha_{p+1}$ où $\alpha_1, \dots, \alpha_p, \alpha_{p+1} \in E^*$.

3. $L(G') \subset L(G) - \{\varepsilon\}$.
   Puisque G' est sans ε-règle, la chaîne vide n'est pas élément de $L(G')$, aussi il suffit de montrer, par induction sur i, que :
   $\boxed{\text{Pour tout non-terminal } A \text{ et toute chaîne terminale } x, \text{ si } A \Rightarrow_{G'}^{i} x \text{ alors } A \Rightarrow_{G}^{*} x}$
   Pour $i = 0$, la propriété est "trivialement" vraie, car il est impossible que $A = x$.
   Supposons $i > 0$ et admettons l'hypothèse d'induction, que la propriété est vérifiée pour toute dérivation de longueur inférieure à i.
   Supposons $A \Rightarrow_{G'}^{i} x$. Puisque $i > 0$, cette dérivation s'écrit :
   $A \Rightarrow_{G'} B_1 \ldots B_p \Rightarrow_{G'}^{i-1} x$ où $B_1, \ldots B_p \in V$.
   En utilisant la propriété fondamentale des langages hors-contexte (voir chapitre III, théorème 1.1), ainsi que l'hypothèse d'induction, il existe des chaînes terminales $x_1, \ldots x_p$ telles que
   – $x = x_1 \ldots x_p$
   – pour j de 1 à p, $B_j \Rightarrow_{G}^{*} x_j$
   Par définition de G', il existe $\alpha_1, \ldots, \alpha_p, \alpha_{p+1} \in E^*$ telles que $A \rightarrow \alpha_1 B_1 \ldots \alpha_p B_p \alpha_{p+1}$ est une règle de G.
   Puisque E est l'ensemble des non-terminaux produisant la chaîne vide dans la grammaire G, toutes les chaînes $\alpha_1, \ldots \alpha_p, \alpha_{p+1}$ produisent la chaîne vide dans cette grammaire et par composition des dérivations, on a : $A \Rightarrow_{G}^{*} x$.

4. $L(G) - \{\varepsilon\} \subset L(G')$.
   Il suffit de montrer,par induction sur i, que :
   $\boxed{\text{Pour tout non-terminal } A \text{ et toute chaîne terminale non vide } x, \text{ si } A \Rightarrow_{G}^{i} x \text{ alors } A \Rightarrow_{G'}^{*} x}$
   Pour $i = 0$, la propriété est "trivialement" vraie.
   Supposons $i > 0$ et admettons l'hypothèse d'induction, que la propriété est vérifiée pour toute dérivation de longueur inférieure à i.
   Supposons $A \Rightarrow_{G}^{i} x$. Puisque $i > 0$, cette dérivation s'écrit :
   $A \Rightarrow_{G} B_1 \ldots B_p \Rightarrow_{G}^{i-1} x$ où $B_1, \ldots B_p \in V$.
   D'après la propriété fondamentale des langages hors-contexte, il existe pour j de 1 à p, des entiers n_j et des chaînes x_j tel que :
   (*)$B_j \Rightarrow_{G}^{n_j} x_j$ et $x = x_1 \ldots x_p$ et $i - 1 = n_1 + \ldots + n_p$.
   Soient $k_1, \ldots, k_q$ la suite croissante des indices des chaînes x_j non vides.
   Par définition des règles de G' et de ces indices :
   (**) $A \rightarrow B_{k_1} \ldots B_{k_q}$ est une règle de G',
   obtenue en supprimant les non-terminaux B_j qui engendrent des chaînes vides.
   Puisque, d'après la propriété **(*)**, pour tout j de 1 à p, $n_j < i$, il résulte, en appliquant l'hypothèse d'induction à cette même propriété que :
   (***) Pour j de 1 à q, $B_{k_j} \Rightarrow_{G'}^{*} x_{k_j}$
   D'après les propriétés **(*)**,**(**)** et **(***)**, en composant les dérivations, on a : $A \Rightarrow_{G'}^{*} x$.
   Par induction la propriété encadrée ci-dessus est donc vérifiée pour tout i.

□

Exemple 3.1 *Soit la grammaire de terminaux a, b, de non-terminaux S, A, B, C, d'axiome S et de règles :*
$S \rightarrow ABC, A \rightarrow BB|\varepsilon, B \rightarrow CC|a, C \rightarrow AA|b$
Tous les non-terminaux produisent la chaîne vide, donc la grammaire équivalente sans ε-règles a comme règles :

$$\begin{array}{lcl} S & \rightarrow & ABC|BC|AC|AB|A|B|C \\ A & \rightarrow & BB|B \\ B & \rightarrow & CC|C|a \\ C & \rightarrow & AA|A|b \end{array}$$

Cet exemple met en évidence que l'enlèvement des règles produisant la chaîne vide peut augmenter considérablement le nombre de règles.

Théorème 3.2 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. Si $L(G)$ n'est pas vide, on peut construire une grammaire hors-contexte $G' =< V_T, V_N, S, R' >$ réduite et sans ε-règle telle que $L(G') = L(G) - \{\varepsilon\}$.*

Preuve : On élimine d'abord les ε-règles, comme il a été indiqué ci-dessus, on obtient alors une grammaire $G_1 =< V_T, V_N, S, R_1 >$ sans ε-règle telle que $L(G_1) = L(G) - \{\varepsilon\}$.
On élimine ensuite les non-terminaux improductifs puis les non-terminaux inaccessibles. Ces éliminations ne faisant qu'enlever des règles, la grammaire obtenue $G_2 =< V_T, V_N, S, R_2 >$ est réduite, équivalente à G_1 et sans ε-règle.
Par suite en posant $R' = R_2$, $G' = G_2$, la grammaire $G' =< V_T, V_N, S, R' >$ est réduite et vérifie $L(G') = L(G) - \{\varepsilon\}$. □

4 Élimination des 1-règles

Rappelons qu'une 1-règle est une règle de la forme $A \to B$ où B est un non-terminal. L'élimination n'a plus qu'un intérêt historique et doit être vu comme un exercice sur les transformations de grammaire.

Théorème 4.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. On peut construire une grammaire hors-contexte équivalente sans 1-règle $G' =< V_T, V_N, S, R' >$.*

Preuve : On montre comment construire G', ensuite on prouve $L(G') \subset L(G)$ puis $L(G) \subset L(G')$.

1. Construction de G'.
   Soit r la relation sur V_N définie par : ArB si et seulement si $A \to B$ est une règle de G.
   On a donné au paragraphe 2.2, l'algorithme de Roy-Warshall qui permet de calculer facilement r^* la fermeture transitive et reflexive de r.
   On pose : $R' = \{A \to \alpha \mid \text{ il existe } B \to \alpha \in R \text{ et } \alpha \notin V_N \text{ et } Ar^*B\}$.
   Puisque r^* est calculable, il en de même de R'.
2. $L(G') \subset L(G)$.
   On donne uniquement l'argument essentiel de la preuve, toute règle de R' peut être remplacée par une dérivation dans G, c'est-à-dire si $A \to \alpha \in R'$ alors $A \Rightarrow_G^* \alpha$.
3. $L(G) \subset L(G')$.
   On prouve par induction sur i que :
   $\boxed{\text{si } A \text{ est un non-terminal et } x \text{ une chaîne terminale telle que } A \Rightarrow_G^i x \text{ alors } A \Rightarrow_{G'}^* x.}$
   Par hypothèse d'induction, supposons la propriété vraie pour toute dérivation de longueur inférieure à i.
   Supposons $A \Rightarrow_G^i x$. Il y a une dérivation de A en x, qui s'écrit $\alpha_0, \ldots \alpha_i$ où $\alpha_0 = A$ et $\alpha_i = x$.
   Puisque A est un non-terminal et que x est une chaîne terminale, il y a un indice j entre 0 et $i-1$ tel que $\alpha_0, \ldots \alpha_j$ sont des non-terminaux et que la chaîne α_{j+1} n'est pas un non-terminal.
   Par définition de R', $A \to \alpha_{j+1}$ et une règle de R'.
   Puisque $\alpha_0, \ldots \alpha_i$ est une dérivation de la grammaire G, on a $\alpha_{j+1} \Rightarrow_G^{i-(j+1)} x$.
   Puisque $i-(j+1) < i$, en appliquant la propriété fondamentale des grammaires hors-contexte et l'hypothèse d'induction, on obtient : $\alpha_{j+1} \Rightarrow_{G'}^* x$.
   Par suite $A \Rightarrow_{G'}^* x$.
   Donc la propriété encadrée ci-dessus est vraie pour tout i, d'où $L(G) \subset L(G')$.

□

Exemple 4.1 *Soit la grammaire de terminaux $+$, $*$, (,), de non-terminaux E, T, F, d'axiome E et de règles*

$$\begin{array}{lcl} E & \to & E+T|T \\ T & \to & T*F|F \\ F & \to & (E)|a \end{array}$$

On a : $Er^*E, Er^*T, Er^*F, Tr^*T, Tr^*F, Fr^*F$.
Par suite les règles de la grammaire équivalente sans 1-règle sont

$$\begin{array}{lcl} E & \to & E+T|T*F|(E)|a \\ T & \to & T*F|(E)|a \\ F & \to & (E)|a \end{array}$$

On constate que cette grammaire a bien plus de règles.
Appelons expression un mot engendré par E, terme un mot engendré par T et facteur un mot engendré par F.
La nouvelle grammaire ne met plus en évidence ni que tout terme est une expression , ni que tout facteur est un terme.

Théorème 4.2 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte. Si $L(G)$ n'est pas vide, on peut construire une grammaire hors-contexte $G' =< V_T, V_N, S, R' >$ résuite, sans ε-règle et sans 1-règle telle que $L(G') = L(G) - \{\varepsilon\}$. Un grammaire ayant ces propriétés est appelée une grammaire* propre.

Preuve : Il suffit d'appliquer *dans l'ordre* les transformations suivantes : élimination des ε-règles, élimination des 1-règles, élimination des improductifs, élimination des inaccessibles.
En effet puisque l'élimination des 1-règles n'ajoute pas de nouvelle partie droite de règle, la grammaire construite par la deuxième élimination n'a pas ε-règles. Il en est de même des deux dernières éliminations : la grammaire finale G' est donc propre.
Puisque ces différentes éliminations change une grammaire en une autre grammaire équivalente à ε près, on a bien $L(G') = L(G) - \{\varepsilon\}$. □

5 Forme normale de Chomsky

La forme normale de Chmosky n'a plus qu'un intérêt historique. Aussi comme le paragraphe précédent, il faut voir ce paragraphe comme l'exposé de méthodes pour prouver l'équivalence de grammaires.

Définition 5.1 (Forme Normale de Chomsky) *Une grammaire hors-contexte est sous forme normale de Chomsky si toutes ses règles sont de l'une des deux formes suivantes :*
- *$A \to BC$ où A, B, C sont des non-terminaux*
- *$A \to a$ où a est un terminal*

Théorème 5.1 *Soit G une grammaire hors-contexte. On peut construire une grammaire G' sous forme normale de Chomsky telle que $L(G') = L(G) - \{\varepsilon\}$.*

Preuve : D'après le théorème 4.2, pour toute grammaire hors-contexte, on peut construire une grammaire propre équivalente à ε près. On peut donc supposer que la grammaire G est sans ε-règle et sans 1-règle.
Posons $G =< V_T, V_N, S, R >$ et soit $V = V_T \cup V_N$.
La grammaire $G' =< V_T, V'_N, [S], R' >$ est construite ainsi
- Les symboles de V'_N sont obtenus en mettant entre crochets les éléments de V et les suffixes de longueur au moins deux des parties droites de règles de R , autrement dit :
  $V'_N = \{[B] \mid B \in V\} \cup \{[\alpha] \mid$ il existe β tel que $|\alpha| \geq 2, A \to \beta\alpha \in R\}$
- Les règles de R' sont

$$\begin{aligned} R' \quad = \quad & \{[A] \to [B_1][B_2 \ldots B_p] \mid A \to B_1 B_2 \ldots B_p \in R,\ B_1, B_2, \ldots, B_p \in V,\ p \geq 2\} \\ & \cup \{[B_1 B_2 \ldots B_p] \to [B_1][B_2 \ldots B_p] \mid [B_1 B_2 \ldots B_p] \in V'_N, p \geq 2\} \\ & \cup \{[a] \to a \mid a \in V_T\} \\ & \cup \{[A] \to a \mid A \to a \in R \text{ avec } a \in V_T\} \end{aligned}$$

La grammaire G' est sous forme normale de Chomsky. Montrons qu'elle est équivalente à G.

1. Montrons que $L(G) \subset L(G')$.
   Il suffit de prouver, par induction sur i, que :
   $\boxed{\text{si } A \text{ est un non-terminal et } x \text{ une chaîne terminale tel que } A \Rightarrow_G^i x \text{ alors } [A] \Rightarrow_{G'}^* x.}$
   Pour $i = 0$, c'est trivial.
   Pour $i > 0$, supposons la propriété vraie pour toute dérivation de longueur inférieure à i.
   Supposons $A \Rightarrow_G^i x$. Puisque $i > 0$, cette dérivation s'écrit :
   $A \Rightarrow_G B_1 \ldots B_p \Rightarrow_G^{i-1} x$ où $B_1, \ldots B_p \in V$.
   D'après la propriété fondamentale des langages hors-contexte, il existe pour j de 1 à p, des entiers n_j et des chaînes x_j tel que :
   (*)$B_j \Rightarrow_G^{n_j} x_j$ et $x = x_1 \ldots x_p$ et $i - 1 = n_1 + \ldots + n_p$.
   On considère deux cas suivant la valeur de p.
   (a) $p = 1$.
   Vu que G ne comporte pas de 1-règle, B_1 est un terminal égal à x. Donc $[A] \to x$ est une règle de G' donc $[A]$ engendre x (en un pas) dans la grammaire G'.

(b) $p > 1$.
Par définition de R', il est clair que $[A] \Rightarrow^*_{G'} [B_1][B_2]\dots[B_p]$, d'ailleurs en exactement $p-1$ pas.
On distingue deux cas suivant que B_i (où $1 \leq i \leq p$) est un terminal ou un non-terminal.

i. si B_i est un terminal alors $B_i = x_i$ et, par définition des nouvelles règles, $[B_i] \to x_i \in R'$, donc $[B_i] \Rightarrow^*_{G'} x_i$.

ii. si B_i est un non-terminal, puisqu'il engendre x_i en moins de i pas, par hypothèse d'induction, on a aussi $[B_i] \Rightarrow^*_{G'} x_i$.

Par composition de ces dérivations, $[B_1][B_2]\dots[B_p] \Rightarrow^*_{G'} x_1x_2\dots x_n$.
Puisque $x = x_1\dots x_p$ et que $[A] \Rightarrow^*_{G'} [B_1][B_2]\dots[B_p]$, il en résulte que $[A] \Rightarrow^*_{G'} x$.

2. Montrons que $L(G') \subset L(G)$.
Nous présentons une preuve informelle comportant seulement l'argument essentiel.
Supposons que $[S]$ engendre la chaîne terminale x dans G'. Il y a une dérivation de G', de x à partir de $[S]$. En *effaçant les crochets* de cette dérivation, on obtient une dérivation (avec des répétitions) de x (qui n'a pas de crochets) à partir de S dans la grammaire G.
Soyons plus formel. Soit $h : V_{N'} \mapsto V*$ l'application qui efface les crochets, autrement dit :
Pour tout $\alpha \in V^*$ tel que $[\alpha] \in V'_N$ alors $h([\alpha]) = \alpha$.
Soit $V' = V_T \cup V_{N'}$. L'application h est étendue en un homomorphisme pour la concaténation de V'^* dans V^* en posant
– pour tout $a \in V_T$, $h(a) = a$
– pour tous $x, y \in V'^*$, $h(xy) = h(x)h(y)$
En examinant tous les règles de R', on voit que si $[\alpha] \to \beta$ est une règle de R' alors soit $h([\alpha]) = h(\beta)$, soit $h([\alpha]) \to \beta$ est une règle de R. On en déduit facilement que $L(G') \subset L(G)$.

□

Exemple 5.1 *On montre la transformation de grammaire ci-dessus. Soit la grammaire de terminaux a, b, de non-terminaux S,A,B, d'axiome S et de règles*

$$\begin{array}{lcl} S & \to & aAB|BBBA \\ A & \to & BAB|a \\ B & \to & AS|b \end{array}$$

La nouvelle grammaire équivalente a comme non-terminaux $[a],[b],[S],[A],[B]$, $[AB],[BBA],[BA]$, *comme axiome* $[S]$ *et comme règles*

$$\begin{array}{lcl} [S] & \to & [a][AB] \mid [B][BBA] \\ [a] & \to & a \\ [AB] & \to & [A][B] \\ [BBA] & \to & [B][BA] \\ [BA] & \to & [B][A] \\ [A] & \to & [B][AB] \mid a \\ [B] & \to & [A][S] \mid b \end{array}$$

Remarquez que le non-terminal $[b]$ *est inutilisé.*

6 Grammaires hors-contexte récursives

Définition 6.1 *Soit* $G = V_T, V_N, S, R >$ *une grammaire hors-contexte et* $V = V_T \cup V_N$.
Un non-terminal A est
– récursif *si et seulement si il existe* $\alpha, \beta \in V^*$ *tel que* $A \Rightarrow^+ \alpha A \beta$.
– récursif à gauche *si et seulement si il existe* $\alpha \in V^*$ *tel que* $A \Rightarrow^+ A\alpha$.
– récursif à droite *si et seulement si il existe* $\alpha \in V^*$ *tel que* $A \Rightarrow^+ \alpha A$.

Une grammaire hors-contexte est *récursive* (respectivement *récursive à gauche*, *récursive à droite* si elle contient un non-terminal récursif (respectivement récursif à gauche, récursif à droite).

Propriété 6.1 *Le problème « la grammaire G est-elle récursive ?» est décidable.*

Preuve : Il suffit de montrer que le problème «le non-terminal A de la grammaire G est-il récursif ?» est décidable.
Soit d la relation de dépendance sur V_N définie au paragraphe 2.2.
Il est clair que A est récursif si et seulement si Ad^+A où d est la fermeture transitive de cette relation de dépendance.
Dans le paragraphe 2.2, on a vu comme calculer d^*, la fermeture transitive et réflexive de d. La calcul de d^+,la fermeture transitive de d, est analogue, il suffit de modifier la définition de d_0 par :
$A_i d_0 A_j$ si et seulement si $A_i d A_j$. □

Les deux paragraphes suivants, on montre que pour toute grammaire hors-contexte, on peut construire une grammaire sans symbole récursif à gauche. Cette transformation est appelée *l'élimination de la récursivité à gauche*. Il a plusieurs façons d'effectuer cette élimination. Nous avons choisi une méthode élémentaire (mais qui n'est pas la meilleure), effectuer une suite d'éliminations de règles *directement récursives à gauche*, c'est-à-dire de règles de la forme $A \to A\alpha$. L'élimination de la récursivité gauche (respectivement droite) est encore actuellement une transformation importante, car les analyseurs descendants (respectivement ascendants) exigent des grammaires non-recursives à gauche (respectivement non-récursives à droites).

7 Élimination des règles directement récursive à gauche

Définition 7.1 *Un non-terminal A est directement récursif à gauche dans une grammaire si la grammaire comporte une règle de la forme $A \to A\alpha$.*

Théorème 7.1 *Soit $G = V_T, V_N, S, R >$ une grammaire hors-contexte et $V = V_T \cup V_N$.*
Soit A un non-terminal directement récursif à gauche et soit A' un nouveau non-terminal, autrement dit $A' \notin V$.
Soit $G' = V_T, V_N \cup \{A'\}, S, R' >$ la grammaire où l'ensemble R' est obtenu en remplaçant les A-règles (c'est-à-dire les règles d'entête A) par
$\{A \to \beta \mid A \to \beta \in R$ *et* β *ne commence pas par* $A\}$
$\cup\{A \to \beta A' \mid A \to \beta \in R$ *et* β *ne commence pas par* $A\}$
$\cup\{A' \to \alpha \mid A \to A\alpha \in R\}$
$\cup\{A' \to \alpha A' \mid A \to A\alpha \in R\}$.
On a les résultats suivants :
- *Les deux grammaires sont équivalentes.*
- *A n'est pas directement récursif à gauche dans G'.*
- *Si G ne comporte pas la règle $A \to A$, alors A' n'est pas directement récursif à gauche dans G'.*

Preuve : Il est évident, par construction de G', que, dans la nouvelle grammaire, A n'est pas récursif à gauche et A' non plus, à condition que $A \to A$ ne soit pas une règle de G.
On montre que $L(G) \subset L(G')$, en donnant uniquement l'argument essentiel, et on laisse au lecteur la preuve (analogue) de l'inclusion inverse.
Dans le début d'une dérivation gauche à partir de A d'une chaîne terminale il y a un premier pas qui ne commence pas par A, donc il existe $p \geq 0$ tel que le début de cette dérivation s'écrit :
Pour i de 1 à p, $A\alpha_{i-1}\ldots\alpha_1 \Rightarrow A\alpha_i\alpha_{i-1}\ldots\alpha_1$ (par la règle $A \to A\alpha_i$ de G)
$A\alpha_p\alpha_{p-1}\ldots\alpha_1 \Rightarrow \beta\alpha_p\ldots\alpha_1$ par la règle $A \to \beta$ où β ne commence pas par A.
Vu le remplacement des récursions directes gauches par des récursions directes droites, on a aussi dans G' :
$A \Rightarrow^*_{G'} \beta\alpha_p\ldots\alpha_1$. En effet :

1. Si $p = 0$, alors $A \to \beta$, est une règle commune à G et G'.
2. Si $p > 0$, alors dans G' il y a la dérivation droite :
   $A \Rightarrow \beta A'$
   Pour i de p à 2, $\beta\alpha_p\ldots\alpha_{i+1}A' \Rightarrow \beta\alpha_p\ldots\alpha_{i+1}\alpha_i A'$ (par la règle $A' \to \alpha_i A'$ de R')
   $\beta\alpha_p\ldots\alpha_2 A' \Rightarrow \beta\alpha_p\ldots\alpha_2\alpha_1$ (par la règle $A' \to \alpha_1$ de R')

On en déduit que si le non-terminal A engendre une chaîne terminale x par dérivation gauche dans G, alors A engendre x dans G'.

D'après la propriété 2.2 du chapitre III, si le non-terminal A engendre une chaîne terminale x dans G, alors il engendre aussi x par dérivation gauche.
Par suite toute chaîne terminale engendrée par un non-terminal de G est aussi engendrée par ce même non-terminal dans G'. Donc $L(G) \subset L(G')$.
On prouve de façon analogue, en remplaçant les dérivations gauches par des dérivations droites, que $L(G') \subset L(G)$. □

Exemple 7.1 *Soit la grammaire de terminaux a,b, de non-terminal S et de règles :*
$S \rightarrow SS|aSb|\varepsilon$.
En éliminant la récursion gauche directe, on obtient une grammaire d'axiome S et de règles
$S \rightarrow aSbS'|S'|aSb|\varepsilon$
$S' \rightarrow SS'|S$
La récursion directe gauche a été éliminée, mais les non-terminaux S et S' sont récursifs à gauche.

8 Élimination de la récursivité à gauche

On élimine la récursion à gauche, car les analyseurs descendants (ceux qui analysent les mots en essayant de construire une dérivation à partir de l'axiome) exigent des grammaires non récursives à gauches.
Nous donnons ci-dessous deux exemples modèles où l'élimination de la récursion gauche est effectuée simplement en la remplaçant par une récursion droite.

Exemple 8.1 *Rappelons la grammaire des expressions arithmiques.*
$E \rightarrow E+T|T$
$T \rightarrow T*F|F$
$F \rightarrow (E)|a$
On peut éliminer la récursion à gauche simplement en la remplaçant par
$E \rightarrow T+E|T$
$T \rightarrow F*T|F$
$F \rightarrow (E)|a$
La nouvelle grammaire, sans récursion gauche, est bien plus courte et plus descriptive que celle résultant des méthodes ci-dessous.

Exemple 8.2 *La grammaire suivante, décrivant des listes de a*
$L \rightarrow La|\varepsilon$
est équivalente à la grammaire sans récursion gauche
$L \rightarrow aL|\varepsilon$
Cet exemple montre que, contrairement aux méthodes ci-dessous, l'élimination des ε-règles n'est pas un préalable indispensable à l'élimination de la récursion à gauche.

Nous proposons ci-dessous un algorithme d'élimination de la récursion gauche, qui opère sur les grammaires sans ε-règle et sans 1-règle. Il existe d'autres algorithmes[1], qui n'obligent pas à respecter ces deux conditions et introduisent moins de non-terminaux en faisant l'analogue de l'élimination directe à gauche simultanément sur plusieurs non-terminaux : l'intérêt de ces algorithmes plus complexes est notamment de mieux respecter les qualités «descriptives» des grammaires, comme sur les exemples ci-dessus.

Algorithme 8.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte sans ε-règle et sans 1-règle. Soit $V = V_T \cup V_N$ et $A_1, \ldots, A_n$ une liste sans répétition des non-terminaux de G.*
On construit une suite $G_{(i|1 \leq i \leq n+1)}$ de grammaires ainsi définies.
$G_1 = G$
Soit $G_i =< V_T, V_{N_i}, S, R_i >$. On montre comment, pour $i \leq n$, on obtient G_{i+1} à partir de G_i.
La construction comporte deux étapes

[1] Voir par exemple la thèse de Jérome Bordier
Méthode pour la mise au point de grammaires LL(1)
Université Joseph-Fourier - Grenoble I (1971-01-30), Jean-Claude Boussard (Dir.)
http://tel.archives-ouvertes.fr/docs/00/04/81/34/PDF/tel-00009479.pdf)

1. *Pour j de 1 à i on constuit la suite de grammaires* $G_{i,j} =< V_T, V_{N_i}, S, R_{i,j} >$.
   $G_{i,1} = G_i$.
   Pour $j < i$, *la grammaire* $G_{i,j+1}$ *est obtenu en remplaçant, chaque règle de* $G_{i,j}$ *de la forme* $A_i \to A_j\alpha$ *par l'ensemble de règles* $\{A_i \to \beta\alpha \mid A_j \to \beta$ *est une règle de* $G_{i,j}\}$.
2. *Si* A_i *n'est pas directement récursif à gauche dans* $G_{i,i}$ *alors* $G_{i+1} = G_{i,i}$.
3. *Si* A_i *est directement récursif à gauche dans* $G_{i,i}$ *alors* G_{i+1} *est construite ainsi :*
   Soit A'_i *un nouveau non-terminal :* $V_{N_{i+1}} =, V_{N_i} \cup \{A'_i\}$.
   On remplace les A_i*-règles par l'ensemble des règles*
   $\{A_i \to \beta \mid A_i \to \beta \in R_{i,i}$ *et* β *ne commence pas par* $A_i\}$
   $\cup\{A_i \to \beta A'_i \mid A_i \to \beta \in R_{i,i}$ *et* β *ne commence pas par* $A_i\}$
   $\cup\{A'_i \to \alpha \mid A_i \to A_i\alpha \in R_{i,i}\}$
   $\cup\{A'_i \to \alpha A'_i \mid A_i \to A_i\alpha \in R_{i,i}\}$.

Exemple 8.3 *Soit la grammaire* G_1 *de terminaux* a, b, c, d, e, *de non-terminaux* A_1,A_2 *et de règles*

$A_1 \quad \to \quad A_2a|b$
$A_2 \quad \to \quad A_2c|A_1d|e$

$G_2 = G_1$, *car* A_1 *n'est pas directement récursif.*
$G_{2,2}$ *a comme règle :*

$A_1 \quad \to \quad A_2a|b$
$A_2 \quad \to \quad A_2c|A_2ad|bd|e$ *En remplaçant* A_1 *par sa définition*

La grammaire G_3 *ci-dessous est obtenue par élimination de la récursion gauche directe sur* A_2 *:*

$A_1 \quad \to \quad A_2a|b$
$A_2 \quad \to \quad bd|e|bdA'_2|eA'_2$
$A'_2 \quad \to \quad cA'_2|adA'_2|c|ad$

Théorème 8.1 *Soit G une grammaire sans* ε*-règle et sans 1-règle. On peut construire une grammaire équivalente à G non récursive à gauche.*

Preuve : On adopte les notations de l'algorithme 8.1.
Il est évident que toutes les grammaires construites sont équivalentes. En effet les transformations de grammaires utilisées sont des substitutions générales et des éliminations de la récursion directe gauche, et les théorèmes 1.2 et 7.1 montrent que ces transformations changent une grammaire en une autre équivalente.
Nous montrons, par induction, que la grammaire G_{n+1} n'est pas récursive à gauche.
Soit $P(i, j)$ la propriété suivante de la grammaire $G(i, j)$:

- Toutes les A_k-règles où $1 \leq k \leq n$ ont des parties droites commençant par un terminal, ou de la forme $A_l\alpha$ où α a un premier symbole élément de V.
- Toutes les A_k-règles où $1 \leq k < i$ ont des parties droites commençant par un terminal ou de la forme $A_l\alpha$ où $k < l$.
- Toutes les règles de la forme $A_i \to A_k\alpha$ où sont telles que $k \geq j$.
- La partie droite de *toutes les règles d'entête un nouveau non-terminal*, commencent par un élément de V.

Nous montrons que pour tout i, j, $P(i, j)$ est vraie.

1. Si $1 \leq i \leq n$ et $j < i$ et si $P(i, j)$ est vraie alors $P(i, j+1)$ est vraie.
   Supposons que $1 \leq i \leq n$, $j < i$ et que la propriété $P(i, j)$ est vérifiée.
   Puisque la grammaire $G(i, j+1)$, est obtenue en remplaçant toutes règles de la forme $A_i \to A_j\alpha$ par $\{A_i \to \beta\alpha \mid A_j \to \beta \in R_{i,j}\}$, et que, d'après la propriété $P(i, j)$, toutes les chaînes β commencent par un terminal ou s'écrivent $A_k\gamma$ où $k > j$ et γ commence par un symbole de V, alors tous les nouvelles règles de $G(i, j+1)$ sont de la forme $A_i \to \delta$ où δ commence par un terminal ou est de la forme $A_k\eta$ où $k \geq j+1$ et η commence par un symbole de V.
   Par suite la propriété $P(i, j+1)$ est vérifiée.
2. Puisque $G_{1,1} = G_1 = G$, et que G est sans ε-règle et sans 1-règle, alors la propriété $P(1, 1)$ est vérifiée.
3. Pour $1 \leq i \leq n$ alors $P(i, 1)$ implique $P(i+1, 1)$.
   Supposons que $1 \leq i \leq n$ et que la propriété $P(i, 1)$ vérifiée.

Puisque $P(i,1)$ est vraie et que pour $j < i$, $P(i,j)$ implique $P(i,j+1)$, alors par induction sur j, $P(i,i)$ est vraie. On examine deux cas suivant que le non-terminal A_i est ou n'est pas directement récursif à gauche dans $G_{i,i}$.

(a) Supposons que A_i n'est pas directement récursif à gauche.
Par définition $G_{i+1,1} = G_{i+1} = G_{i,i}$.
Puisque la propriété $P(i,i)$ est vérifiée et, vu l'absence de récursion directe gauche sur A_i, toutes les A_i-règles où $1 \leq k < i$ ont des parties droites commençant par un terminal ou de la forme $A_k\alpha$ où $i < k$. Donc la propriété $P(i+1,1)$ est vérifiée.

(b) Supposons que A_i est directement récursif à gauche.
La grammaire $G_{i+1,1} = G_{i+1}$ est obtenu par élimination de la récursion directe gauche, autrement dit les A_i règles de $R_{i,i}$ sont remplacées par l'ensemble de règles
$\{A_i \to \beta \mid A_i \to \beta \in R_{i,i}$ et β ne commence pas par $A_i\}$
$\cup\{A_i \to \beta A_i' \mid A_i \to \beta \in R_{i,i}$ et β ne commence pas par $A_i\}$
$\cup\{A_i' \to \alpha \mid A_i \to A_i\alpha \in R_{i,i}\}$
$\cup\{A_i' \to \alpha A_i' \mid A_i \to A_i\alpha \in R_{i,i}\}$.
D'après la propriété $P(i,i)$, les chaînes α commencent par un élément de V, donc toutes A_i' règles commencent par un élément de V.
D'après la propriété $P(i,i)$, les chaînes β commencent par un terminal ou s'écrivent $A_j\gamma$ où $j > i$ et γ commence par un symbole de V.
Donc tous les nouvelles règles de $G_{i+1} = G(i+1,1)$ sont de la forme $A_i \to \delta$ où δ commence par un terminal ou sont de la forme $A_k\eta$ où $k \geq j+1$ et η commence par un symbole de V.
Par suite $P(i+1,1)$ est aussi vérifiée.

4. La grammaire G_{n+1} n'est pas récursive à gauche.
On a vu ci-dessus que $P(1,1)$ est vraie et que pour tout $1 \leq i \leq n$, $P(i,1)$ implique $P(i+1,1)$. Donc par induction la propriété $P(n+1,1)$ est vérifiée.
Par suite la grammaire G_{n+1}, où $G_{n+1} = G_{n+1,1}$ vérifie :

(a) Toute partie droite d'un A_i-règle (où $1 \leq i \leq n$) commençe par un terminal ou par A_j où $i < j$.

(b) Tout nouveau non-terminal a une partie droite commençant par un élément de V

De ces deux propriétés, il résulte que G_n n'est pas récursive à gauche. En effet
- Supposons que $A_i \Rightarrow^+ \alpha$.
Par induction sur la longueur de cette dérivation, α commence que par un terminal ou un non-terminal A_j où $j > i$. Donc les non-terminaux A_i ne sont pas récursifs à gauche.
- Supposons que $A_i' \Rightarrow^+ \alpha$.
Puisque les parties droite des A_i'-règles commencent par un élément de B, il existe un élément $B \in V$ et des chaînes β, γ, δ telles que :
* $A_i' \Rightarrow B\beta \Rightarrow^* \gamma\delta$
* $B \Rightarrow^* \gamma$ et $\beta \Rightarrow^* \delta$ et $\alpha = \gamma\delta$
Puisque B engendre ou est égal à γ, alors d'après ce qui vient d'être démontré, γ, donc aussi α, commence par un symbole de V. Donc les non-terminaux A_i' ne sont pas récursifs à gauche.

Par suite la grammaire G_{n+1} n'est pas récursive à gauche.

□

Notons un conséquence immédiate du théorème. Toute grammaire est équivalente au mot vide près à une grammaire non récursive à gauche. Il suffit d'éliminer dans l'ordre les ε-règles, les 1-règles, puis d'appliquer l'algorithme 8.1 ci-dessus.

9 Forme normale de Greibach

Définition 9.1 *Une grammaire est sous la forme normale de Greibach si toutes ses règles sont de la forme $A \to aB_1 \ldots B_p$ où $p \geq 0$, a est un terminal et les B_i sont des non-terminaux.*

Algorithme 9.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte sans ε-règle, et non récursive à gauche. L'algorithme transforme G en une grammaire équivalente dont toutes les parties droites de règles commencent par un terminal.*
Considérons la relation g suivante entre les symboles de V_N :
$A\ g\ B$ si et seulement si il existe une chaîne α telle que $A \Rightarrow^+ B\alpha$.
Cette relation est un ordre partiel : en effet, elle est évidemment transitive, et, puisque la grammaire n'est pas récursive à gauche, elle est irréflexive.
Par suite il existe un ordre total[2], c'est-à-dire une liste sans répétition des symboles de V_N, $A_1, \ldots, A_n$ prolongeant l'ordre partiel c'est à dire tel que si $A_i\ g\ A_j$ alors $i < j$.
Si $A_i \rightarrow A_j\alpha$ est une règle de G, alors $A_i\ g\ A_j$ donc $j > i$. Puisque G ne comporte pas d'ε-règles, les autres règles de G commencent par un terminal.
Une fois numérotés les non-terminaux dans l'ordre indiqué, on construit par des substitutions générales une grammaire dont toutes les règles commence par un terminal.
Pour i de n à 1, on construit la suite de grammaires $G_i =< V_T, V_N, S, R_i >$ de la façon suivante :

- *$G_n = G$*
- *pour $n \geq i \geq 2$ la grammaire G_{i-1} est obtenue en remplaçant chaque règle de G_i de la forme $A_{i-1} \rightarrow A_j\alpha$ par $\{A_{i-1} \rightarrow \beta\alpha \mid A_j \rightarrow \beta \in R_i\}$.*

Théorème 9.1 *Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte sans ε-règle, sans 1-règle, non récursive à gauche. Soit $V = V_T \cup V_N$. On peut construire une grammaire équivalente dont toutes les parties droites de règle commencent par un terminal.*

Preuve : On adopte les notations de l'algorithme 9.1.
Il est évident que toutes les grammaires construites sont équivalentes. En effet les transformations de grammaires utilisées sont des substitutions générales et le théorème 1.2 montre que ces transformations changent une grammaire en une autre équivalente.
Nous montrons que toutes les parties droites des règles de la grammaire G_1 commencent par un terminal.

Soit $P(i)$ la propriété suivante de la grammaire G_i : (a) pour tout $j < i$, toutes les parties droites de A_j-règles commencent par un terminal ou sont de la forme $A_k\alpha$ où $k > j$. (b) pour $i \leq j$, toutes les parties droites des A_j-règles commencent par un terminal.

1. $P(n)$ est vérifiée.
   D'après l'ordre adopté des non-terminaux, toutes les règles de G commencent par un terminal ou sont de la forme $A_i \rightarrow A_j\alpha$ où $j > i$.
   Comme il y a n non-terminaux, les A_n règles commencent par un terminal.
   Puisque $G_n = G$, $P(n)$ est vérifiée.
2. Pour $i \geq 2$, $P(i)$ implique $P(i-1)$.
   Supposons $i \geq 2$ et la propriété $P(i)$ vérifiée.
   Toute A_{i-1}-règle de G_i qui ne commence pas par un terminal, commence par un non-terminal A_j où $i \leq j$. Par construction, A_j est remplacé par les parties droites des A_j-règles et d'après $P(i)$, ces parties droites commencent par un terminal.
   Donc après ces remplacements, la propriété $P(i-1)$ est vérifiée.
3. Par induction puisque $P(n)$ est vérifiée et puisque, pour tout $i \geq 2$,$P(i)$ implique $P(i-1)$, il résulte que $P(1)$ est vérifiée. Donc tous les parties droites de règles commencent par un terminal.

□

Théorème 9.2 *Pour toute grammaire hors-contexte, on peut construire une grammaire équivalente à ε près sous forme normale de Greibach.*

Preuve : Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte et $V = V_T \cup V_N$.
Soit $A_1, \ldots, A_n$ une liste sans répétitions des non-terminaux de cette grammaire.
La construction de la grammaire sous forme normale de Greibach, est faite en plusieurs étapes dans l'ordre ci-dessous :

[2] Cet ordre est calculable par l'algorithme du tri topologique

1. Élimination des ε-règles décrite au paragraphe 3.

2. Élimination des 1-règles décrite au paragraphe 4.
   Puisque l'élimination des 1-règles n'introduit pas d'ε-règles, la grammaire obtenue après cette étape, est sans ε-règle et sans 1-règle.
3. Élimination de la récursion à gauche par l'algorithme 8.1.
   Comme résultat de cet algorithme, on obtient une grammaire $G' =< V_T, V_{N'}, S, R' >$ avec $V_{N'} = V_N \cup E$ ou E est un ensemble de nouveaux non-terminaux et les règles de R' ayant les propriétés suivantes :
   (a) La partie droite des règles ayant pour entête un nouveau non-terminal commence par un élément de V
   (b) Toute partie droite d'une A_i règles (où $1 \leq i \leq n$) commencent par un terminal ou par un non-terminal A_j où $j > i$.

   Soit $B_1, \ldots, B_p$ une liste sans répétition des nouveaux non-terminaux. Il en résulte que l'ordre total utilisé dans l'algorithme 9.1 n'a pas besoin d'être calculé, il peut être défini par la liste $B_1, \ldots, B_p, A_1, \ldots, A_n$ dans laquelle tout non terminal A précéde tout non-terminal B tel que $A \; g \; B$ (c'est-à-dire tels que il existe une chaîne α telle que $A \Rightarrow^+ B\alpha$).
4. Transformation de la grammaire G' en une grammaire dont tous les parties droites de règle commencent par un terminal. Il suffit d'appliquer l'algorithme 9.1, avec l'ordre total des non-terminaux déjà défini ci-dessus.
5. Passage à la forme normale de Greibach. Il suffit de :
   (a) Remplacer chaque règle de la forme $A \rightarrow a\alpha$ par $A \rightarrow a\alpha'$ où α' est obtenu en remplaçant chaque terminal b par un nouveau non-terminal b'.
   Par exemple la règle $A \rightarrow aBbaC$ où a et b sont des terminaux est remplacée par la règle $A \rightarrow aBb'a'C$ où a' etb' sont des nouveaux non-terminaux (avec un nouveau non-terminal pour chaque terminal).
   (b) Pour chaque terminal a, ajouter la règle $a' \rightarrow a$

   Il est évident que cette transformation change une grammaire en une autre équivalente.

□

Chapitre IX

Conditions d'unicité des systèmes d'équation

Ce chapitre est un complément du chapitre qui utilise les systèmes d'équation pour donner des définitions inductives des langages hors-contextes. Nous nous intéressons aux conditions nécessaires et suffisantes *calculables* pour qu'un système d'équation ait une seule solution.
Ce chapitre est un hommage à Bruno Courcelle, voir http://www.labri.fr/perso/courcell/, qui est l'auteur de cette étude intéressante et trop souvent oubliée.
Dans le premier paragraphe, on définit un système strict et on montre qu'un tel système a une seule solution. Dans le deuxième paragraphe, on étudie des conditions nécessaires et suffisantes pour qu'un système ait une seule solution : un système riche [1] a une seule solution si et et seulement si on peut le transformer par substitutions en un système strict et cette condition est calculable.

1 Système strict

Définition 1.1 (système strict) *Le définition d'une variable d'un système est stricte si tous ses mots sont vides ou comportent un terminal. Un système d'équations est strict si toutes les définitions de ses variables sont strictes.*

Exemple 1.1 *Le système $S = aSb + \varepsilon$ est strict.*
Le système $S = SS + aSb + \varepsilon$ n'est pas strict.

Théorème 1.1 *Un système strict a une seule solution.*

Preuve : La preuve n'est qu'une généralisation de l'exemple étudié au paragraphe 1 du chapitre IX.
Soit $S =< V_T, V_N, d >$ un système strict d'équations et $V = V_T \cup V_n$.
Soit I la plus petite solution du système et J une solution quelconque. Supposons que ces deux solutions soient distinctes.
Il existe x une chaine terminale *de longueur minimale* et un non terminal A tel que $x \in \mathrm{J}(A)$ et $x \notin \mathrm{I}(A)$, autrement dit, pour toute chaine terminale y de longueur inférieure à x et tout non terminal B, $x \in \mathrm{J}(B)$ si et seulement si $x \in \mathrm{I}(B)$.
Donc il existe α un mot de la définition de A tel que $x \in \mathrm{J}(\alpha)$ et $x \notin \mathrm{I}(\alpha)$.
Le mot α n'est pas une chaine terminale, car dans ce cas, nous aurions $\mathrm{I}(\alpha) = \mathrm{J}(\alpha) = \{\alpha\}$. Donc $\alpha = B_1 B_2 \ldots B_n$ où $B_1, B_2, \ldots B_n \in V$, $n \geq 1$ et, puisque le système est strict, au moins un de ces symboles est un terminal et un autre un non terminal.
Par définition des interprétations, il existe une suite $x_1, x_2, \ldots x_n$ de chaines terminales telles que, pour i de 1 à n, $x_i \in \mathrm{J}(B_i)$ et $x = x_1 x_2 \ldots x_n$.
Soit B_i est un terminal. Alors $\mathrm{J}(B_i) = \mathrm{I}(B_i) = \{x_i\}$

[1] Un système riche est défini en 2.1

Soit B_i est un non terminal. Puisque que le système est strict, une des chaines $x_1, x_2, \ldots x_n$ est un symbole terminal, donc la longueur de x_i est inférieure (strictement) à celle de x. Et, vu que la chaine x est la chaine la plus courte pour laquelle existe un non terminal A avec $x \in \mathrm{J}(A)$ et $x \notin \mathrm{I}(A)$, on a $x_i \in \mathrm{I}(B_i)$.
Par suite $x \in \mathrm{I}(\alpha)$, donc $x \in \mathrm{I}(A)$: on obtient une contradiction. Donc $I = J$ et le système n'a qu'une solution. □

2 Conditions nécessaires et suffisantes d'unicité

Soit S un système d'équations, transformé par des substitutions générales [2] en un système strict S'. D'après la propriété 5.2 du chapitre IX, toute solution de S est une solution de S'. D'après le théorème 1.1, un système strict n'a qu'une solution. Donc S n'a qu'une solution.

Exemple 2.1 *Le système d'équations S suivant n'est pas strict :*
$E = T + E \underline{+} T$
$T = F + T * F$
$F = a + b + (E)$
Cependant on peut le transformer en un système strict S' par des substitutions générales :
$E = a + b + (E) + T * F + E \underline{+} T$
$T = a + b + (E) + T * F$
$F = a + b + (E)$
Donc S n'a qu'une solution.

On peut se demander si le fait pour un système d'être transformable par des substitutions générales en un système strict est une condition nécessaire et suffisante d'unicité des solutions. Un exemple nous montre que ce n'est pas tout à fait le cas.
Soit le système $S = SS + a + \varepsilon$. Par des substitutions générales, il est impossible de transformer ce système en un système strict. Lorsque le vocabulaire terminal ne comprend que la lettre a, l'unique solution de ce système est $S = a^*$. Par contre lorsque le vocabulaire terminal comprend les deux lettres a et b, le système a les deux solutions $S = a^*$ et $S = (a+b)^*$.

Définition 2.1 (Système riche) *Un systeme d'équations est riche, si et seulement si, l'ensemble des mots de la plus petite solution du système est* strictement *inclus dans l'ensemble des chaines sur le vocabulaire terminal du système.*

Lorsque l'on considère le système $S = SS + a + \varepsilon$ comme ayant le vocabulaire terminal $\{a\}$, il n'est pas riche. Mais il est riche, si l'on considère que son vocabulaire terminal est $\{a, b\}$. On montre dans la suite qu'un système *riche* a une seule solution si et seulement si il est transformable par des substitutions générales en un système strict.

Définition 2.2 (Variables potentiellement strictes) *Soit $S =< V_T, V_N, d >$ un système d'équations et $V = V_T \cup V_N$. Soit E l'ensemble de variables engendrant la chaine vide au moyen des règles du système. On a vu, au paragraphe 3 du chapitre VIII, comment calculer cet ensemble.*
Soit $f : \mathcal{P}(V_N) \mapsto \mathcal{P}(V_N)$ la fonction suivante :
pour tout $X \subset V_N$, $f(X) = \{A \in V_N \mid \forall \alpha \in d(A), \alpha \in X^* \cup V^* V_T V^* \cup V^*(X - E)V^*\}$.
La fonction f est une fonction croissante sur le treillis fini $\mathcal{P}(V_N)$ ordonné par l'inclusion. Puisqu'on sait calculer E, la fonction f est calculable. En suivant l'exemple du calcul de l'ensemble des symboles productifs donné au paragraphe 2.1 du chapitre VIII, cette fonction a un plus petit point fixe F calculable *ainsi :*
Soit n le nombre de variables du système, $F = f^i(\emptyset)$ où i est le plus petit entier inférieur à n tel que $f^i(\emptyset) = f^{i+1}(\emptyset)$.
F est l'ensemble des variables potentiellement strictes du système : ce nom vient, comme on le voit ci-dessous, de ce que F comporte les variables strictes du système ainsi que les variables qui peuvent devenir strictes par substitution.

Lemme 2.1 *Soit S un système d'équations et F l'ensemble des variables potentiellement strictes du système. Si F est l'ensemble des variables de S, alors S est transformable par substitutions en un système strict.*

[2] On rappelle qu'une telle substitution consiste à remplacer dans les définitions du système, des occurrences de variables par leur définition

Preuve : Soit $S =< V_T, V_N, d >$ un système d'équations et $V = V_T \cup V_N$.
Soit F l'ensemble des variables potentiellement strictes du système.
Soit $f : \mathcal{P}(V_N) \mapsto \mathcal{P}(V_N)$ la fonction suivante :
pour tout $X \subset V_N$, $f(X) = \{A \in V_N \mid \forall \alpha \in d(A), \alpha \in X^* \cup V^* V_T V^* \cup V^*(X-E)V^*\}$.
On sait que F est le plus petit point fixe de f et qu'il existe un entier i tel que $F = f^i(\emptyset)$.
Pour tout entier j, posons $F_j = f^j(\emptyset)$.
Soit $S_O = S$ et S_{i+1} le système ainsi obtenu : dans les définitions de S_i, on remplace *toutes* les occurrences de variables par leur définitions dans S_i.
Montrons que :
(*) pour tout entier j, l'ensemble F_j est inclus dans l'ensemble des variables strictes du système S_j.
Pour $j = 0$ c'est évident puisque F_0 est l'ensemble vide.
Supposons que F_j est inclus dans l'ensemble des variables strictes du système S_j et montrons qu'il en est de même en remplaçant j par $j+1$.
Soit $A \in F_{j+1}$. On examine deux cas.

1. Supposons $A \in F_j$. Par hypothèse d'induction, A est strict dans S_j. Puisque S_{j+1} est obtenu par substitution sur S_j, la variable A reste stricte dans S_{j+1}.
2. Supposons $A \notin F_j$. Par définition de F_{j+1}, toutes les règles $A \to \alpha$ sont telles que :
   $\alpha \in F_i^* \cup V^* V_T V^* \cup V^*(F_i - E)V^*$.
   Soit $A \to \beta$ une règle de S_{j+1}, obtenue en remplaçant dans α les variables par leurs définitions dans S_j. Montrons que β est vide ou comporte un terminal. On examine trois cas suivant l'ensemble dont α est élément.
   (a) Soit $\alpha \in F_i^*$. Par hypothèse de récurrence toute variable de α est stricte dans S_i, donc est remplacée par la chaine vide ou des chaines comportant un terminal. Par suite β est vide ou comporte un terminal.
   (b) Soit $\alpha \in V^* V_T V^*$. Alors β comporte un terminal, puisqu'il y en a déjà un dans α.
   (c) Soit $\alpha \in V^*(F_i - E)V^*$. Donc α comporte une variable $X \in F_i - E$. Par récurrence cette variable est stricte dans S_j, donc les mots de sa définition sont vides ou comprennent un terminal. Puisque cette variable ne produit pas la chaine vide, ses parties droites de règles comportent toutes un terminal, qui, en remplaçant X par sa définition dans S_j, se retrouve dans β.

   Donc la variable A devient stricte dans S_{j+1}.

Supposons que F est l'ensemble des variables du système S.
On sait qu'il existe un entier i tel que $F = F_i$. D'après la propriété **(*)**, pour tout j, F_j est inclus dans l'ensemble des variables strictes de S_j.
Puisque l'ensemble des variables ne change pas, toutes les variables de S_i sont strictes. Donc le système S_i, obtenu en transformant S par des substitutions, est strict. □

Lemme 2.2 *Soit $S =< V_T, V_N, d >$ un système d'équations et F l'ensemble des variables potentiellement strictes du système. Soit $\overline{F} = V_N - F$ l'ensemble complémentaire de F. Pour toute variable A, on a :*

1. *$A \in F$ si et seulement si pour tout $\alpha \in d(A)$, on a : $\alpha \in F^* \cup V^* V_T V^* \cup V^*(F-E)V^*$.*
2. *$A \in \overline{F}$ si et seulement si il existe une chaine $\alpha \in d(A)$, telle que : $\alpha \in (E \cup \overline{F})^* \overline{F} (E \cup \overline{F})^*$*

Preuve : Soit $f : \mathcal{P}(V_N) \mapsto \mathcal{P}(V_N)$ la fonction suivante :
pour tout $X \subset V_N$, $f(X) = \{A \in V_N \mid \forall \alpha \in d(A), \alpha \in X^* \cup V^* V_T V^* \cup V^*(X-E)V^*\}$.
On sait que F est le plus petit point fixe de f, donc $f(F) = F$, ce qui est équivalent à la première propriété.
La négation de la condition $\alpha \in F^* \cup V^* V_T V^* \cup V^*(F-E)V^*$ est la conjonction des 3 propriétés

(a) $\alpha \notin F^*$, autrement dit un des symboles de α n'est pas dans F, c'est-à-dire $\alpha \in V^* \overline{F} V^*$.
(b) $\alpha \notin V^* V_T V^*$, autrement dit tous les symboles de α sont des variables, c'est-à-dire $\alpha \in V_N^*$.
(c) $\alpha \notin V^*(F-E)V^*$, autrement dit tous les symboles de α sont éléments de $\overline{F-E}$ où $\overline{F-E} = \overline{F \cap \overline{E}} = E \cup \overline{F}$

Puisque tous les éléments de $E \cup \overline{F}$ sont des variables, la condition (c) implique la condition (b), et donc la conjonction des propriétés (a, b, c) est équivalente à la conjonction de la propriété (b) «un des symboles de α est élément de $\overline{F}$» et de la propriété (c), «tous les éléments de α sont éléments de $E \cup \overline{F}$».
Par suite la négation de la condition $\alpha \in F^* \cup V^* V_T V^* \cup V^*(F-E)V^*$ est $\alpha \in (E \cup \overline{F})^* \overline{F} (E \cup \overline{F})^*$.

Donc la propriété 2 est obtenue en remplaçant dans la propriété 1 les deux membres de l'équivalence par leur négations : elle est donc équivalente à la propriété 1. □

Lemme 2.3 *Soit S un système riche d'équations (voir définition 2.1). Soit F l'ensemble des variables potentiellement strictes du système. Si F n'est pas l'ensemble de toutes les variables du système, alors S a plusieurs solutions.*

Preuve : Soit $S =< V_T, V_N, d >$ et I la plus petite solution de S. Puisque S est riche, il y a une chaine terminale w qui n'appartient pas à $\bigcup\{I(A) \mid A \in V_N\}$.
Soit $\overline{F} = V_N - F$, le complément de F et J l'interprétation suivante définie pour toute variable A par

- si $A \in \overline{F}$ alors $J(A) = \{\varepsilon, w\}$
- si $A \in E \cap F$ alors $J(A) = \{\varepsilon\}$.
  On rappelle que E est l'ensemble des variables engendrant ε grâce aux règles du système.
- si $A \in F - E$ alors $J(A) = \emptyset$.

Soit f_S la fonction définie par le système S (voir la définition 3.4 du chapitre IX). On montre tout d'abord que $J \subset f_S(J)$, c'est-à-dire que pour tout $A \in V_N$, $J(A) \subset f_S(J)(A)$.
On examine trois cas suivant la définition de J :

1. Soit $A \in \overline{F}$. $J(A) = \{\varepsilon, w\}$.
   D'après le cas 2 du lemme 2.2, il existe une chaine $\alpha \in d(A)$ telle que $\alpha \in (E \cup \overline{F})^* \overline{F} (E \cup \overline{F})^*$.
   Notons que tous les symboles de α sont des variables. Soit B une telle variable.
   Puisque $B \in E \cup \overline{F}$ et que $E \cup \overline{F} = (E \cap F) \cup \overline{F}$, d'après les deux premiers cas de la définition de J, on a $\varepsilon \in J(B)$.
   D'autre part il y a symbole C de α tel que $C \in \overline{F}$, donc par définition de J, $J(C) = \{\varepsilon, w\}$.
   Par suite $\{\varepsilon, w\} \subset J(\alpha)$, donc $J(A) \subset f_S(J)(A)$.
2. Soit $A \in E \cap F$. $J(A) = \{\varepsilon\}$.
   Puique $A \in E$, il existe $\alpha \in d(A) \cap E^*$. D'après le cas 1 du lemme 2.2, on a nécessairement $\alpha \in F^*$. Donc toutes les variables B de α sont éléments de $E \cap F$, et, par définition de J, $J(B) = \{\varepsilon\}$.
   Par suite $\{\varepsilon\} \subset J(\alpha)$, donc $J(A) \subset f_S(J)(A)$.
3. Soit $A \in F - E$. $J(A) = \emptyset$ donc $J(A) \subset f_S(J)(A)$.

Soit alors l'interprétation $K = \bigcup\{f_S^i(J) \mid i \geq 0\}$.
Puisque $I \subset f_S(J)$ et que f_S est une fonction continue (voir le théorème 3.1 du chapitre IX), on déduit, en remplaçant dans la preuve du théorème de Kleene (voir 2.1 du chapitre IX) l'élément $\perp$ par J, que K est un point fixe de f_S, c'est-à-dire une solution du système S.
Par hypothèse, $\overline{F}$ n'est pas vide. Soit $A \in \overline{F}$. De ce que $J \subset K$, on déduit que $w \in K(A)$. Puisque $w \notin \bigcup\{I(A) \mid A \in V_N\}$, on sait que $w \notin I(A)$.
Donc le système S a deux solutions distinctes. □

Théorème 2.1 *Soit S un système d'équations qui est riche et F l'ensemble de ses variables potentiellement strictes. Les trois propriétés suivantes sont équivalentes :*

(1) S n'a qu'une solution.
(2) F est l'ensemble des variables de S
(3) S est transformable par substitution en un système strict

Preuve : (1) implique (2) : c'est la contraposée du lemme ci-dessus 2.3.
(2) implique (3) : c'est le lemme 2.1.
(3) implique (1) : Supposons que S est transformable par substitutions en un système strict S'. D'après le théorème 5.2 du chapitre IX, toute solution de S est solution de S'. D'après le théorème 1.1, le système strict S' n'a qu'une solution. Donc S n'a qu'une solution. □

3 Problèmes de décision

Un problème est décidable, s'il y a un algorithme pour le résoudre. Nous renvoyons à l'ouvrage [5] pour plus de précisions sur ces notions.

3.1 Le problème «un système est-il riche ?» est indécidable

3.1.1 Système riche : rappel

Soit $S =< V_T, V_N, d >$ un système et $V = V_T \cup V_N$. V_N est l'ensemble des variables et $d : V_N \mapsto \mathcal{P}(V)$ donne la définition de chaque variable.
L'ensemble R_S des règles de S est l'ensemble des règles $A \to \alpha$ telles que $\alpha \in d(A)$.
Soit A une variable. On note $G_{S,A}$ la grammaire $< V_T, V_N, R_S, A >$ et $L(G_{S,A})$ le langage qu'elle engendre.
Le système S est riche si et seulement si $\bigcup\{L(G_{S,A}) \mid A \in V_N\}$ est un sous-ensemble strict de V_T^*.

3.2 Indécidabilité du problème «un système est-il riche ?»

Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte.
Le théorème 8.11 p203 du livre [5], nous dit que le probleme «$L(G) = V_T^*$?» est indécidable.
Nicolas Peltier (`http://membres-liglab.imag.fr/peltier/`) a trouvé une réduction de ce problème au problème «un sytème est-il riche ?» : autrement dit, si le problème «un système est-il riche ?» était décidable, le problème «$L(G) = V_T^*$?» serait décidable.
Comme ce dernier problème n'est pas décidable, il en est de même du problème «un sytème est-il riche ?». On présente ci-dessous les principes de cette réduction.
Soit $G =< V_T, V_N, S, R >$ une grammaire hors-contexte.
À cette grammaire on associe le système $S' =< V'_T, V'_N, d >$ suivant :

1. V'_T est le vocabulaire terminal V_T auquel on ajoute la *nouvelle* lettre e.
2. V'_N est l'ensemble de variables V_N auquel on ajoute les nouvelles variables A, B, C, D, E.
3. Les règles de S' sont les suivantes
   (a) Les règles de R
   (b) $A = eS$.
   Autrement dit A engendre l'ensemble des mots $eL(G)$.
   (c) B engendre l'ensemble des mots ne commençant pas par le *nouveau* terminal e.
   Les règles pour B sont : $B = \varepsilon + \{a|a \in V_T\}(C + CeB + CEB)$
   C engendre V_T^* et E engendre ee^+.
   (d) $C = \varepsilon + \{a|a \in V_T\}C$
   (e) $E = ee + eE$
   (f) D engendre l'ensemble des mots comportant au moins deux e.
   Les règles pour D sont : $D = CeCeB + CeCEB$

   On laisse au lecteur le soin de vérifier, qu'avec les règles ci-dessus, B engendre l'ensemble des mots ne commençant pas par e et D engendre l'ensemble des mots comportant au moins deux e.

Notons $U(S')$, l'union des langages engendrés par chacune des variables de S'.
D'après les propriétés de A, B, C, D, E, on a : $L(G_{S',C}) \subset L(G_{S',B})$ et $L(G_{S',E}) \subset L(G_{S',D})$.
Puisque toutes les variables de la grammaire G engendrent des mots sans e, donc éléments de $L(G_{S',B})$, il en résulte que : $U(S') = L(G_{S',A}) \cup L(G_{S',B}) \cup L(G_{S',D})$.
Montrons que $L(G) = V_T^*$ si et seulement si $U(S') = V_T'^*$ (c'est-à-dire si S' n'est pas riche).

1. Supposons $L(G) = V_T^*$.
   D'après cette hypothèse et le fait que $A = eS$, on a : $U(S') = eV_T^* \cup L(G_{S',B}) \cup L(G_{S',D})$.
   Soit $w \in V_T'^*$.
   (a) $w = \varepsilon$, donc $w \in L(G_{S',B})$ et par suite $w \in U(S')$
   (b) w a une seule occurrence de e. On a deux cas suivant que cette occurrence est ou n'est pas la première lettre de w.
      i. e est la première lettre de w. Donc $w \in eV_T^*$ et par suite $w \in U(S')$.
      ii. e n'est pas la première lettre de w. Donc $w \in L(G_{S',B})$ et par suite $w \in U(S')$.

(c) w a au moins deux occurrrences de e.
Puis que D engendre l'ensemble des mots comportant au moins deux e, $w \in L(G_{S',D})$ et par suite $w \in U(S')$.

Par suite $w \in U(S')$, donc $U(S') = V_T'^*$.

2. Supposons $U(S') = V_T'^*$.
Soit $w \in V_T^*$. Puisque ew n'est engendré ni par B, ni par D, $ew \in eL(G)$. Par suite $w \in L(G)$, donc $L(G) = V_T^*$

3.2.1 Le problème «un système a-t-il une seule solution ?» est relativement décidable

Soit S un système riche. Le problème «S a-t-il une seule solution ?» est décidable puisque, d'après le théorème ci-dessus 2.1, on peut le résoudre en calculant l'ensemble des variables potentiellement strictes de S.
Soit $S =< V_T, V_N, d >$ un système quelconque. Si l'on est pas certain que ce système est riche, on peut le transformer en un système riche, en ajoutant à V_T des lettres qui ne figurent pas dans les règles de S.

Bibliographie

[1] Michael A.Harrison. *Introduction to Formal language theory*. Addison-Wesley, 1978.

[2] Jean-Michel Autebert. *Théorie des langages et des automates*. Masson, 1994.

[3] Walter Bucher and Hermann Maurer. *Theoretische Grundlagen de Programmier-sprachen*. Bibliographisches Institut, 1984.

[4] Seymour Ginsbourg. *The mathematical theory of context-free languages*. MacGraw-Hill, 1979.

[5] John E. Hopcroft and Jeffrey D. Ullmann. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1979.